Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Bachelorthesis

Jonas Radtke

Implementierung des „Server Message Block“-Protokolls auf einer SPS

Fakultät Technik und Informatik
Department Informations- und Elektrotechnik

Faculty of Engineering and Computer Science
Department of Information and Electrical Engineering

Jonas Radtke

Implementierung des „Server Message Block“-Protokolls auf einer SPS

Bachelorthesis eingereicht im Rahmen der Bachelorprüfung
im Studiengang Informations- und Elektrotechnik
am Department Informations- und Elektrotechnik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer : Prof. Dr.-Ing. Ulfert Meiners
Zweitgutachter : Prof. Dr.-Ing. Holger Gräßner

Abgegeben am 12. Februar 2018

Jonas Radtke

Thema der Bachelorthesis

Implementierung des „Server Message Block“-Protokolls auf einer SPS

Stichworte

SMB, Server Message Block, CoDeSys, SPS, Dateiübertragung

Kurzzusammenfassung

Diese Arbeit umfasst die Implementierung des “Server Message Block”-Protokolls auf einer CoDeSys SPS. Dies ermöglicht einen einfachen und flexiblen Dateienaustausch zwischen freigegebenen Ordern auf Windows Rechnern und der SPS. Die Performance wird anschließend mit anderen Möglichkeiten verglichen und die Vor-und Nachteile diskutiert.

Jonas Radtke

Title of the paper

Implementation of „Server Message Block“-Protocol on PLC

Keywords

SMB, Server Message Block, CoDeSys, PLC, File Transfer

Abstract

This report comprised the implementation of Server Message Block protocol on a CoDeSys PLC. This allows easy and flexible file transfer between shared folders on Windows machines and PLC. The performance will be examined and other methods for file transfer on PLC will be discussed.

Inhaltsverzeichnis

Tabellenverzeichnis

Abbildungsverzeichnis

1 Einführung

1.1 Motivation und Zielsetzung

Die Idee zu dieser Arbeit kam mir während meines Praxissemesters. Es gibt immer wieder Anlagen bei denen auf der Steuerung ein Datenlogger implementiert ist oder automatisch Protokolle über geprüfte Waren erstellt werden. Dabei stellt sich immer wieder die Frage wie diese Dateien, meist csv oder txt, zu einem angeschlossenem PC übertragen werden sollen. Dabei handelt es sich meist um Anlagen, welche nicht an ein großes Firmennetzwerk angeschlossen sind, sondern bei denen ein Rechner, auf dem auch die Bedienung mittels WebVisu läuft, direkt daneben steht.

Zwar gibt es schon einige Möglichkeiten zur Übertragung, diese erwiesen sich aber immer wieder als unflexibel oder nicht automatisierbar.

Ziel ist eine Übertragung von Dateien in beide Richtungen, dabei sollen in der Steuerungssoftware die Pfade leicht angepasst werden können. Außerdem soll das ganze ohne zusätzlichen Aufwand auf dem Zielrechner funktionieren. Um dieses Ziel zu erreichen wurde sich für das „Server Message Block“-Protokoll entschieden. Das Ziel ist die Implementierung eines Client-Bausteins für CoDeSys 3.5. Auf dem angeschlossenem Windowsrechner läuft der Server, welcher standardmäßig aktiviert ist. Es muss nur ein Ordner mit Lese- bzw. Schreibberechtigung freigegeben werden.

Minimales Ziel ist das einloggen mit Benutzernamen/Passwort und übertragen von mindestens 1024kb Dateien sowohl lesend als auch schreibend. Auf der SPS werden diese Dateien dann auf dem internen Speicher (z.b. der SD-Karte) abgelegt oder von dort gelesen. Außerdem soll mittels WebVisu ein einfacher Explorer erstellt werden, mit welchem man die freigegeben Ordner und Ordnerinhalte betrachten kann.

1.2 Struktur der Arbeit

Begonnen wird mit den Grundlagen, es wird auf das Netzwerk, die allgemeinen Funktion des Protokolls und den Methoden zur Authentifizierung, bzw. der gewählten Methode, beim

Server eingegangen. In Kapitel 3 wird implementiert, hier wird kurz die verwendete Hardware beschrieben, danach folgt die Erklärung der benutzten Funktionen des Protokolls und deren Implementierung in CoDeSys. Auf die Authentifizierung wird einzeln eingegangen, da dies einen großen Teil einnimmt. In Kapitel 4 wird der Funktionstest beschrieben und anschließend die Performance der Übertragung getestet. Zusätzlich wird dies mit den anderen Übertragungsmöglichkeiten verglichen. In Kapitel 5 wird ein zusammenfassender Überblick und ein Ausblick auf mögliche Verbesserungen gegeben.

2 Grundlagen

In diesem Kapitel werden allgemeine Grundlagen zu den verwendeten Technologien behandelt.

2.1 CoDeSys

CoDeSys ist eine Entwicklungsumgebung für Speicherprogrammierbare Steuerungen und setzt dabei den IEC 61131-3 Standard um. Entwickler ist 3S-Smart Software Solutions GmbH. CoDeSys kann nach Anmeldung kostenlos von der Seite des Herstellers bezogen werden. Sehr viele verschiedene SPS-Hersteller setzten CoDeSys als Entwicklungsumgebung für ihre Steuerungen ein. Lizenzgebühren fallen pro Laufzeitsystem an. Aufgrund der weiten Verbreitung und des kostenlosen Bezugs wurde sich für dieses entschieden.

Laufzeitumgebung
: 3S bietet direkt eine Laufzeitumgebung an, CODESYS Control. Diese wird auch mit der Entwicklungsumgebung direkt ausgeliefert und kann bis zu 2 Stunden kostenfrei benutzt werden. Zusätzlich gibt es eine Laufzeitumgebung für den Raspberry Pi, auch diese läuft 2 Stunden ohne Lizenzierung. Die Implementierung soll anfangs möglichst nicht für einen bestimmten Hersteller entwickelt werden, daher wird die Raspberry Pi Runtime verwendet. Diese ermöglicht es die Software auf einem anderen Gerät als dem Entwicklungsrechner zu testen und ist sehr günstig zu beschaffen.

Visualisierung
: Die Entwicklungsumgebung ermöglicht es eine Visualisierung zu erstellen. Zum debugging kann diese direkt in der Entwicklungsumgebung aufgerufen und auf Variablen zugegriffen werden. Bei manchen Herstellern sowie beim Raspberry Pi ist es möglich damit eine echte Visualisierung zu erstellen. Der Zugriff geschieht mittels Webbrowser und IP Adresse der Steuerung. Dies wird in der Arbeit für den Explorer verwendet.

2.2 Server Message Block - Protokoll

Das Server Message Block Protokoll ist ein Netzwerkprotokoll, es wird für Datei- und Druckdienste benutzt. SMB wurde zuerst von IBM eingeführt und später von verschiedenen Herstellern erweitert, darunter Microsoft. Die Version 2 und 3 sind Erweiterungen des originalen Server Message Block Protokolls. Die Dokumentation dazu stellt Microsoft auf ihrer Website zur Verfügung.

Die Paketformate in der zweiten Version unterscheiden sich stark von der ersten. Durch Reduzierung der Kommandos wurde außerdem die Geschwindigkeit erhöht.

In modernen Windowsversionen wird die Verbindung direkt über TCP/IP Port 445 aufgebaut, ursprünglich wurde NetBios over TCP verwendet. SMB2 verwendet Request/Reponse Kommandos, d.h. der Client schickt einen Request an den Server und dieser antwortet mit einem Response. Ein SMB2 Paket besteht aus einen Transport Header, einem SMB2 Header und dem Request/Reponse Kommando. Auf die verwendeten Header und Kommandos, und deren Strukturen, wird im späteren Verlauf eingegangen.

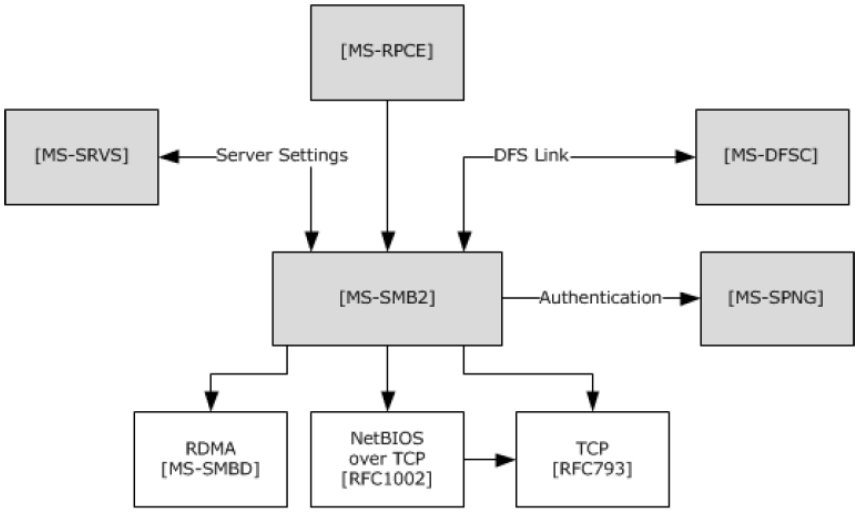


Abbildung 2.1: Verbindung zu anderen Protokollen (Quelle: [5] , S.22)

In Abb. 2.1 wird die Verbindung zu anderen Protokollen dargestellt. Da sind zum einen die drei Transportprotokolle RDMA, NetBios over TCP und TCP. Zum anderen sieht man die Protokolle auf die zurückgegriffen werden. MS-SPNG zur Authentifizierung ist dabei eine API auf Windowsrechnern, diese ist auf der SPS natürlich nicht vorhanden, daher muss diese selbst umgesetzt werden. Einige Kommandos aus den anderen Protokollen müssen für die Funktion von SMB2 implementiert sein.

2.2.1 Header

Der Header, welcher bei jeder Nachricht benötigt wird besteht immer aus zwei Teilen, dem Transport Header und dem SMB2 Header.

Transport Header

Der Transport Header ist 4 Byte lang. Es kennzeichnet den Versand über TCP und enthält die nachfolgende Nachrichtenlänge in Byte. Die Darstellung entspricht der Network Byte Order.

SMB2 Header

Der SMB2 Header wird vor jedem Request oder Response platziert.

Mit jedem Request wird die MessageID um eins erhöht, das Statusfeld mit 0 vorbelegt und Command kennzeichnet den nachfolgenden Request. Bei einem Respone vom Server antwortet dieser mit gleicher MessageID und gleichem Command. Das Statusfeld gibt hierbei an ob der Request erfolgreich war oder welcher Fehler aufgetreten ist.

Die SessionID wird während des Session Setups vergeben, die TreeID nach einem TreeConnect-Request.

Alle Zahlen innerhalb des SMB2 Protokolls werden im Little-Endian Format übertragen.

2.2.2 Verbindungsaufbau

In Abb. 2.2 ist die Ablauf des Verbindungsaufbaus zu einer Freigabe dargestellt. Diese beinhaltet das aushandeln der Verbindungsparameter, die Authentifizierung beim Server und das Verbinden zu einem Ordner.

SMB Negotiate

SMB Negotiate ist ein Befehl aus der ersten SMB Version. Für die allererste Verbindung mit dem Server wird dieser benutzt um den Dialekt auszuhandeln. Dazu wird eine Liste mit allen verfügbaren Versionen angehängt. Bei dieser Arbeit nur ein Eintrag in der Liste. Im Response übermittelt der Server den gewünschten Dialekt, außerdem werden die maximalen Nachrichtengrößen, welche der Server verarbeiten kann übermittelt. Im Anhang des Response wird eine Liste mit möglichen Authentifizierungsmethoden bereitgestellt.

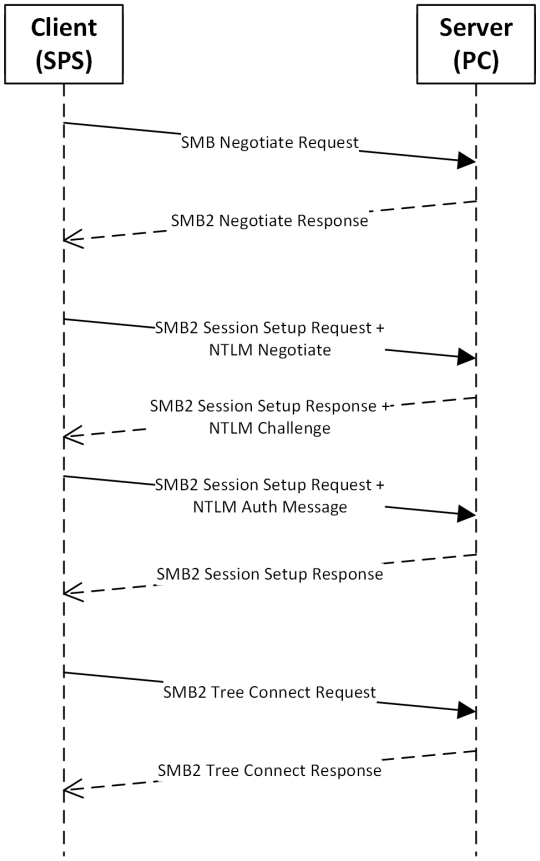


Abbildung 2.2: Verbindungsaufbau zu einer Freigabe

SMB2 Session Setup

Mit dem Session Setup wird sich beim Server eingeloggt. Es wird die gewünschte Authentifizierungsmethode mitgeteilt und die entsprechende Nachricht angehängt. In Kap. 2.3 wird darauf genauer eingegangen. Nach erfolgreicher Authentifizierung erhält der Client vom Server eine SessionID. Fortan wird diese bei jeder Nachricht im Header angegeben.

SMB2 Tree Connect

Mittels Tree Connect verbindet man sich zu einem freigegeben Ordner. Dazu wird in einem Request der Namen übermittelt. Dieser bezieht nur auf den Freigabenamen, nicht auf mögliche Unterordner, diese gehören zum Dateinamen.

Wenn der freigegeben Ordner existiert schickt der Server eine TreeID zurück, andernfalls einen Fehlercode.

2.2.3 Dateiübertragung

Nach erfolgreichem verbinden und erhalten der SessionID und TreeID kann die Übertragung von Daten beginnen. Der Ablauf ist in Abb. 2.3 dargestellt. Hierbei wird zuerst die Datei geöffnet oder angelegt, dabei werden auch die entsprechende Rechte des eingeloggten Benutzers überprüft. Anschließend beginnt die Übertragung von Dateien, abschließend wird die geöffnete Datei geschlossen.

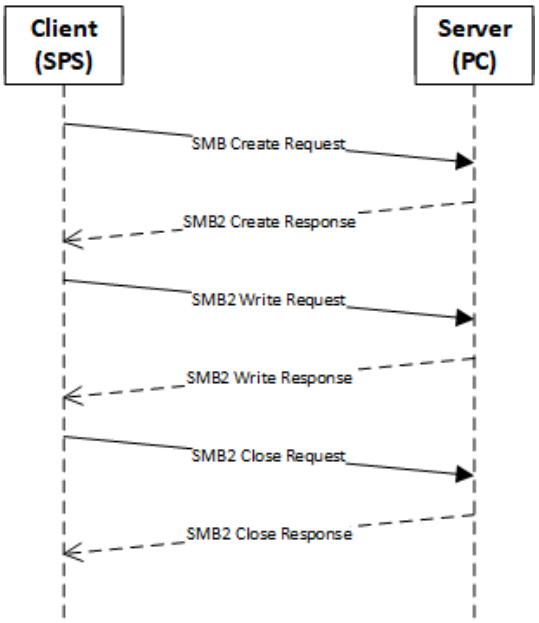


Abbildung 2.3: Schreiben in einen freigegeben Ordner

SMB2 Create

Für den Create Request wird aus dem vorherigen Ablauf die TreeID benötigt. Diese gibt an in welcher Freigabe die gewünschte Datei liegt. Im Request wird der Dateiname angegeben. Dazu gehören auch die vorhergehenden Unterordner, siehe Abb. 3.5. Außerdem werden die gewünschten Optionen zum öffnen angegeben. Dazu gehören lesen/schreiben aber auch was passieren soll wenn betreffende Datei schon existiert, z.b. Fehler oder Überschreiben. Beim Lesen muss die Datei natürlich existent sein. Ist alles richtig gibt der Server im Response eine FileID zurück.

SMB2 Write/Read

Beim Write Request wird die Länge der Daten in der Nachricht, der Dateioffset und die Länge der noch verbleibenden Daten angegeben. Dabei darf die Länge pro Nachricht

nicht der ausgehandelten Länge überschreiten. Bei größeren Datenmengen muss daher der Write Request mehrmals hintereinander ausgeführt werden. Als Payload hängen die Daten einfach hinten dran. Analog dazu gibt es den Read Request, dafür ist nur die zu lesende Länge und Offset innerhalb der Datei zu übermitteln. Beide Befehle benötigen eine gültige FileID, welche beim Create ermittelt wurde.

SMB2 Close

Für einen Close Request wird nur die FileID der zu schliessenden Datei benötigt. Nach dem Response ist die FileID ungültig und kann verworfen werden, für einen erneuten Zugriff muss wieder ein Create Request gesendet werden.

2.2.4 Verbindungsabbau

Bevor die TCP/IP Verbindung getrennt wird muss sich beim Server ordentlich abgemeldet werden. Daher wird sich vom Ordner getrennt und die Session beendet.

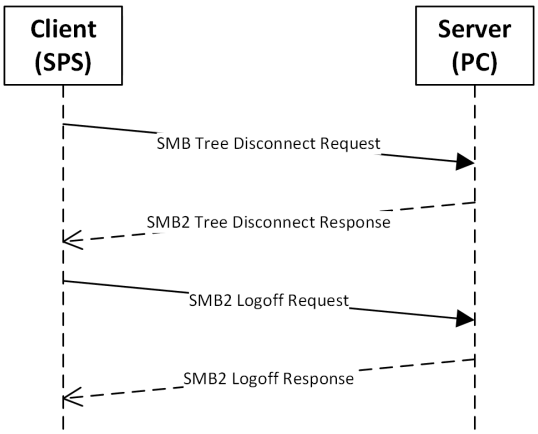


Abbildung 2.4: Verbindungsabbau

SMB2 Tree Disconnect

Hiermit wird sich von einem freigegeben Ordner getrennt. Im Header wird die TreeID angegeben, auf welche sich der Disconnect bezieht. Der Request selbst besteht nur aus seiner Länge und zwei Byte mit 0. Mit dem Response wird das erfolgreiche trennen bestätigt. Die gespeicherte TreeID kann dann verworfen werden. Nach einem Tree Disconnect kann sich mit Tree Connect zum gleichen oder zu anderen freigegeben Ordnern verbunden werden. Andernfalls wird sich ausgeloggt oder auf neue Aktionen vom Benutzer gewartet.

SMB2 Logoff

Mit dem Logoff wird sich komplett abgemeldet. Der Logoff Request ist genauso kurz wie der Tree Disconnect. Geantwortet wird mit dem Logoff Response. Danach kann auch die SessionID verworfen werden. Erneutes verbinden ist nur mit Negotiate und dem Session Setup möglich.

Nach einem Logoff kann die TCP Verbindung ordnungsgemäß getrennt werden.

2.2.5 Anfordern der Ordner- und Dateinamen

Das suchen nach Dateinamen ist relativ einfach, mittels weniger Befehle, gelöst. Nach dem erfolgreichen Verbindungsaufbau kann nach Namen gesucht werden. Siehe Abb. 2.5 mit den aufgerufen Kommandos.

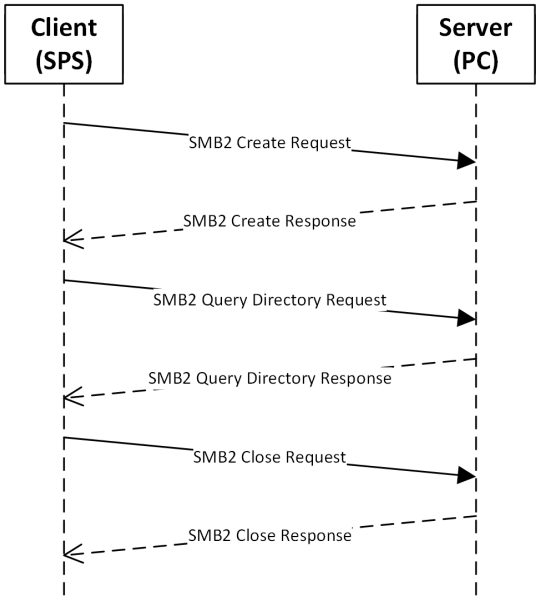


Abbildung 2.5: Suche nach Ordner- und Dateinamen

SMB2 Query Directory

Mit Hilfe dieses Kommandos kann nach beliebigen Ordner bzw. Dateinamen gesucht werden. Zuvor wird mittels Create eine Datei ohne Name geöffnet. Das heißt der Create Request wird normal ausgeführt nur enthält das Feld, welches normalerweise einen Namen beinhaltet, keine Daten. Zusätzlich muss man sich in einem offenen Tree mit gültiger TreeID befinden. Anschließend wird mit Query Directory ein String, nach welchem gesucht werden soll, übergeben. Für eine Suche nach allen Namen wird „*“

gesendet. Falls Namen bzw. Buchstaben übergeben werden wird nur nach diesen gesucht. Der Response liefert eine Liste mit allen gefunden Namen und zusätzlichen Informationen zur Datei, wie z.b. Dateigröße, Datum und Attributen wie Ordner oder Datei.

2.3 Authentifizierungsmethoden

Beim SMB-Protokoll kommen hauptsächlich zwei verschiedene Authentifizierungsverfahren zum Einsatz. Während der Anmeldung zum Server stellt dieser eine Liste zusammen, aus welcher das gewünschte Verfahren ausgewählt werden kann. Da sind zum einen NTLM (NT LAN Manager) und Kerberos V5.

2.3.1 Kerberos V5

Kerberos ist im RFC4120 [2] spezifiziert. Im Gegensatz zu NTLM authentifiziert sich hier zusätzlich der Server gegenüber dem Client. Dies geschieht über eine dritte Partei, dem Kerberos-Server. Der Client meldet sich mit Benutzernamen und Passwort beim Kerberos-Server an und erhält ein Ticket und einen Sitzungsschlüssel, welches er zum Server sendet. Der Server überprüft die Gültigkeit des Ticket beim Kerberos-Server und gewährt dem Client dann Zugang. Der Sitzungsschlüssel kann zusätzlich zur Verschlüsselung benutzt werden. Dieses Verfahren wurde aufgrund des hohen Aufwandes nicht implementiert.

2.3.2 NT LAN Manager

NT LAN Manager ist ein Authentifizierungsverfahren von Microsoft. Spezifiziert sind NTLMv1 und NTLMv2 in einem Microsoft Dokument [4], aus welchem auch die Informationen zu diesem stammen.

NTLM nutzt eine Challenge-Response Authentifizierung. Hierbei wird im Laufe des SMB2 Verbindungsaufbau eine Zufallszahl an den Client gesendet. Dieser bildet aus dem Benutzernamen zusammen mit dem Passwort einen Hashwert. Mit diesem Hashwert wird die Zufallszahl verschlüsselt und zurück gesendet. Der Server hat die entsprechenden Hashwerte vom Benutzername/Passwort gespeichert und verschlüsselt die Zufallszahl ebenfalls mit diesem. Bei Übereinstimmung gewährt dieser dem Client Zugang. NTLM ist immer eine der Angebotenen Authentifizierungsverfahren bei SMB2, ist die Authentifizierung mit Kerberos nicht möglich, wird auf NTLM zurückgegriffen.

Hash- und Verschlüsselungsverfahren

Im folgenden wird auf die Grundlagen der benötigten Verfahren eingegangen.

MD4 - Message-Digest Algorithm 4
MD4 ist eine kryptologische Hashfunktion, welche aus beliebigen Daten einen 16 Byte langen Hashwert bildet. Aus diesem ist die Originalnachricht nichtmehr zurück berechenbar. Eine kleine Änderung der Eingangsdaten ergibt einen völlig anderen Hashwert. Aufgrund von Sicherheitslücken wird MD4 aber nicht mehr empfohlen [8].

MD5 - Message-Digest Algorithm 5
MD5 ist die Weiterentwicklung von MD4, auch diese erzeugt einen 16 Byte Hashwert aus beliebigen Eingangsdaten [9]. MD5 wird in dieser Arbeit außer bei NTLM auch zur Überprüfung der übertragenen Dateien eingesetzt.

HMAC - Keyed-Hash Message Authentication Code
HMAC ist ein Algorithmus zur Nachrichtenauthentifizierung. Dieses nutzt die Nachricht und einen geheimen Schlüssel und berechnet mit diesen einen Hashwert. Dies ist sicherer als die bloße Übertragung des Hashwerts [3]. In dieser Arbeit wird HMAC mit MD5 als Hashfunktion benutzt, es ist aber auch möglich andere Hashalgorithmen zu nutzen.

RC4
RC4 ist ein Stromverschlüsselungsverfahren. Es wird zuerst ein Array von Bytes mittels geheimen Schlüssel initialisiert. Danach kann eine Nachricht damit verschlüsselt werden, genauso funktioniert die Entschlüsselung. Die verschlüsselte Nachricht hat die selbe Länge wie unverschlüsselt.

Authentifizierung

In Abb. 2.6 ist der Ablauf zur Authentifizierung dargestellt. Da NTLM nicht allein funktioniert muss es immer mittels eines anderen Protokolls übermittelt werden. Dies geschieht während des Session Setup bei SMB2, daher sind auch zusätzlich die SMB2 Kommandos dargestellt.

Negotiate Message
Mit der Negotiate Message werden nur die gewünschten Flags übermittelt, welche die Konfiguration angeben. Außerdem wird dem Server so die gewünschte Authentifizierungsmethode übermittelt.

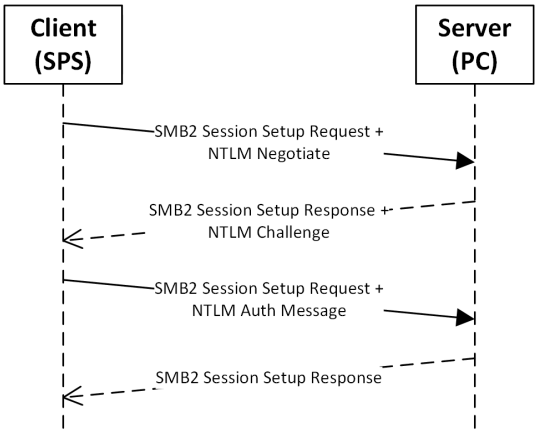


Abbildung 2.6: Authentifizierung am Server

Challenge Message

Die Challenge Message wird vom Server an den Client geschickt. Diese enthält neben der Zufallszahl auch einige Verbindungsattribute. Mittels der genannten Hash- und Verschlüsselungsalgorithmen wird die Zufallszahl verarbeitet.

Auth Message

In die Auth Message werden alle berechneten Werte zurück an den Server geschickt. Dieser überprüft das Ergebnis und gewährt mittels Antwort Zugang.

2.4 Transmission Control Protocol (TCP)

TCP ist ein Internetprotokoll. Es regelt wie Daten zwischen Netzwerkkomponenten ausgetauscht werden. Die meisten heutigen Betriebssysteme unterstützen und nutzen es für den Datenaustausch. Spezifiziert wird es in RFC793[7] und RFC7323[1]. In dieser Arbeit wird die fertige CoDeSys Implementierung des Protokolls benutzt.

TCP ist eine Ende-zu-Ende-Verbindung welches gleichzeitiges Senden und Empfangen erlaubt. Eine Verbindung zwischen zwei Geräten wird durch zwei Endpunkte definiert. Jeder Endpunkt besteht dabei aus IP-Adresse und Port.

Zwischen denselben Geräten können auch mehrere TCP Verbindungen aufgebaut werden, dabei ist es nötig einen anderen Port zu wählen, die IP-Adresse bleibt dabei gleich. Mehrere TCP Verbindungen werden benötigt falls verschiedene Anwendungen kommunizieren möchten.

Die Größe eines TCP-Segments beträgt maximal 1500 Bytes, da es aber meist mittels IP übertragen wird müssen hier 20 Byte IP-Header abgezogen werden. Zusätzlich 20 Byte TCP-Header, macht insgesamt 1460 Byte für Nutzdaten. Größere Datenpakete müssen daher in kleinere Pakete aufgeteilt und separat verschickt werden. Dazu wird jedem Teil ein TCP-Header vorangestellt, darin enthalten eine Segmentnummer. Anhand dieser setzt der Empfänger die Pakete wieder in der richtigen Reihenfolge zusammen.

Für jedes Paket welches versandt wurde, wird ein Retransmission Timer gestartet. Falls ein Paket vom Empfänger nicht quittiert wird, muss dieses nach Ablauf der Zeit noch einmal versandt werden.

3 Implementierung

In diesem Kapitel wird die Softwareimplementierung und der Hardwareaufbau erläutert.

3.1 Hardware

Für die Implementierung musste sich für eine Hardwarelösung entschieden werden. Aufgrund des günstigen Preises und der leichten Beschaffung fiel die Wahl auf einen Raspberry Pi 3. Die Laufzeitumgebung für den Pi stellt 3S zur Verfügung. Diese ist 2 Stunden lauffähig, danach muss neu gestartet werden. Dies ist aber völlig ausreichend um später die Software zu testen. Eine harte Echtzeit wird nicht erreicht, wird aber auch nicht benötigt. Weiterer Vorteil ist, es kann mit der Standard CoDeSys IDE Software entwickelt werden. Als Betriebssystem kommt Raspbian zum Einsatz, dies wird auch von 3S in der Installationsanleitung empfohlen. Nach Einrichtung der des Betriebssystems und Verbindung mit dem Netzwerk wird die Laufzeitumgebung mittels CoDeSys V3.5 installiert. Der Aufbau im Netzwerk ist relativ einfach gehalten, auf einen Windows PC läuft der SMB2 Server und auf der Laufzeitumgebung des Raspberry Pi läuft der entwickelte SMB2 Client, siehe Abb. 3.1.

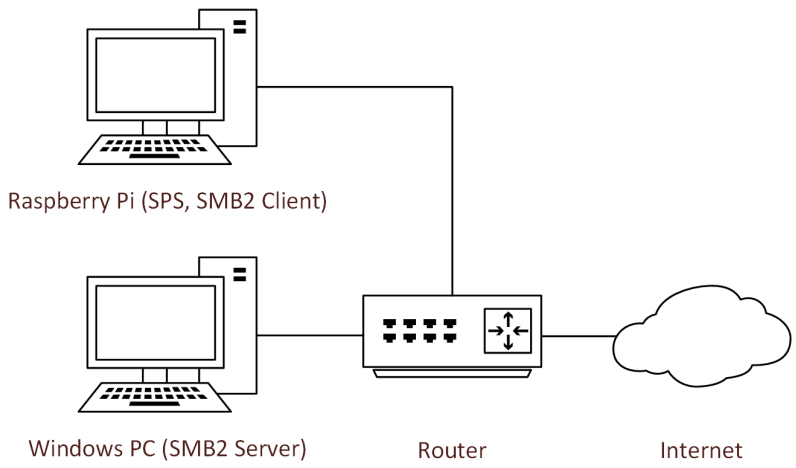


Abbildung 3.1: Übersicht Netzwerkaufbau

3.2 Allgemeiner Aufbau der Software

3.2.1 Projekterstellung

Das gesamte Projekt wurde in CoDeSys V3.5 SP12 erstellt. Folgende Bausteinarten werden von CoDeSys unterstützt.

- Program (PRG) - Ein Program Baustein ist die höchste Ebene. Beim bearbeiten bleiben alle Werte bis zur nächsten Bearbeitung erhalten. Mehrfache Aufrufe sind möglich, er kann allerdings nicht instantiiert werden. Das heißt es gibt ihn nur einmal. Jeder Aufruf ändert Daten in genau diesem einem Baustein. Aus einem Program Baustein können weitere Bausteine aufgerufen werden. Außerdem kann er als einziger Baustein in einem eigenen Task laufen.
- Function (FUN) - Functions können mehrere Eingangsvariablen besitzen, haben aber nur eine Rückgabevariable, welcher bei der Erstellung angegeben wird. Daten in diesem Baustein bleiben nur solange erhalten wie er aufgerufen wird. Diese besitzen beim nächsten Aufruf also wieder ihre Startwerte.
- Function Block (FB) - FBs können mehrere Eingangs- und Ausgangsvariablen besitzen. Anders als FUN speichert dieser Baustein die Daten bis zum nächsten Aufruf. Zusätzlich kann er instantiiert werden, das heißt er kann mehrfach auftauchen und verschiedene Daten enthalten.

Das Program „PLC_PRG“ dient als Ausgangspunkt in einem eigenen Task. Jeder weitere FB oder FUN wird hier aufgerufen. Da die Software möglichst einfach weiter verwendbar sein soll wurde hier nur der „SMB2_Client“-Baustein aufgerufen. Dieser kann später aber in anderen Projekten an jeder Stelle in der Software benutzt werden. Einzige Voraussetzung ist die zyklische Bearbeitung.

In der Software werden zusätzlich verschiedene Bibliotheken benötigt. Diese beinhalten Funktionen, welche vom Hersteller oder Drittanbietern implementiert wurden und in diesem Projekt benutzt werden.

- Standard (System)
- Standard64 (System)
- FileAccess (3S Smart Software Solutions GmbH)
- Memory (3S Smart Software Solutions GmbH)
- Network (3S Smart Software Solutions GmbH)
- Basic (OSCAT)

In der Entwicklungsumgebung wurden die Bausteine und Datenstrukturen in Ordnern gruppiert. In Abb. 3.2 sind die Bausteine und in Abb. 3.3 die Strukturen abgebildet.

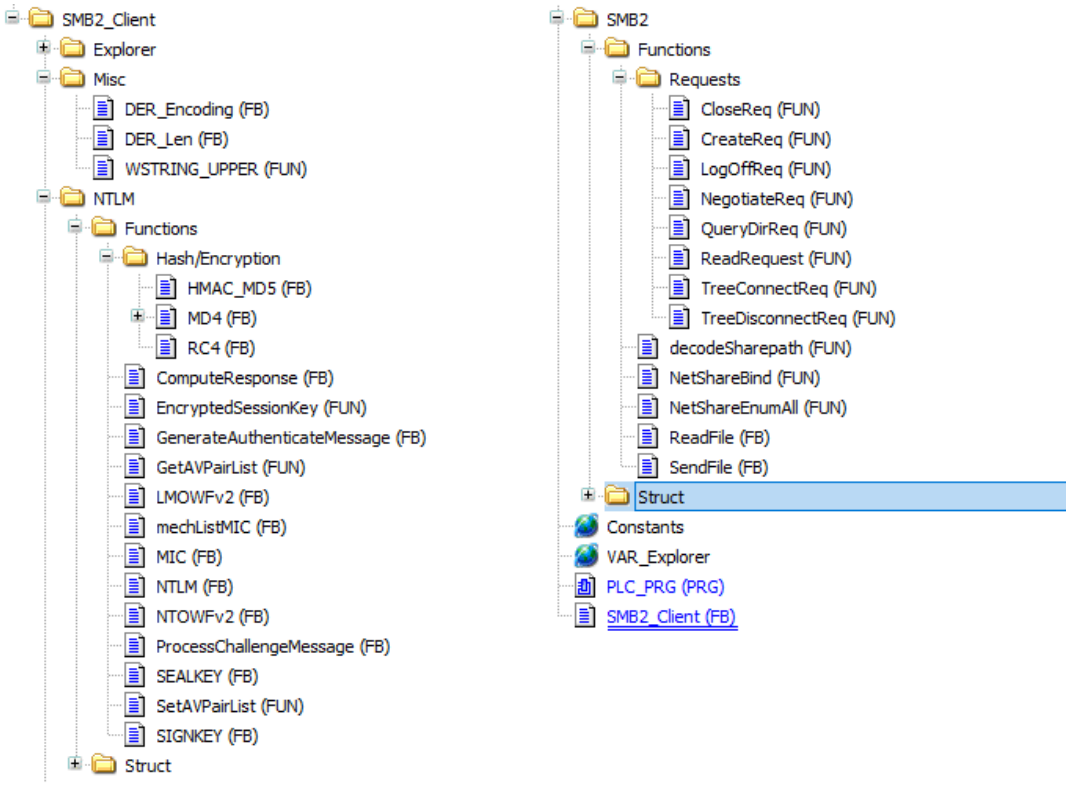


Abbildung 3.2: Bausteingruppierung

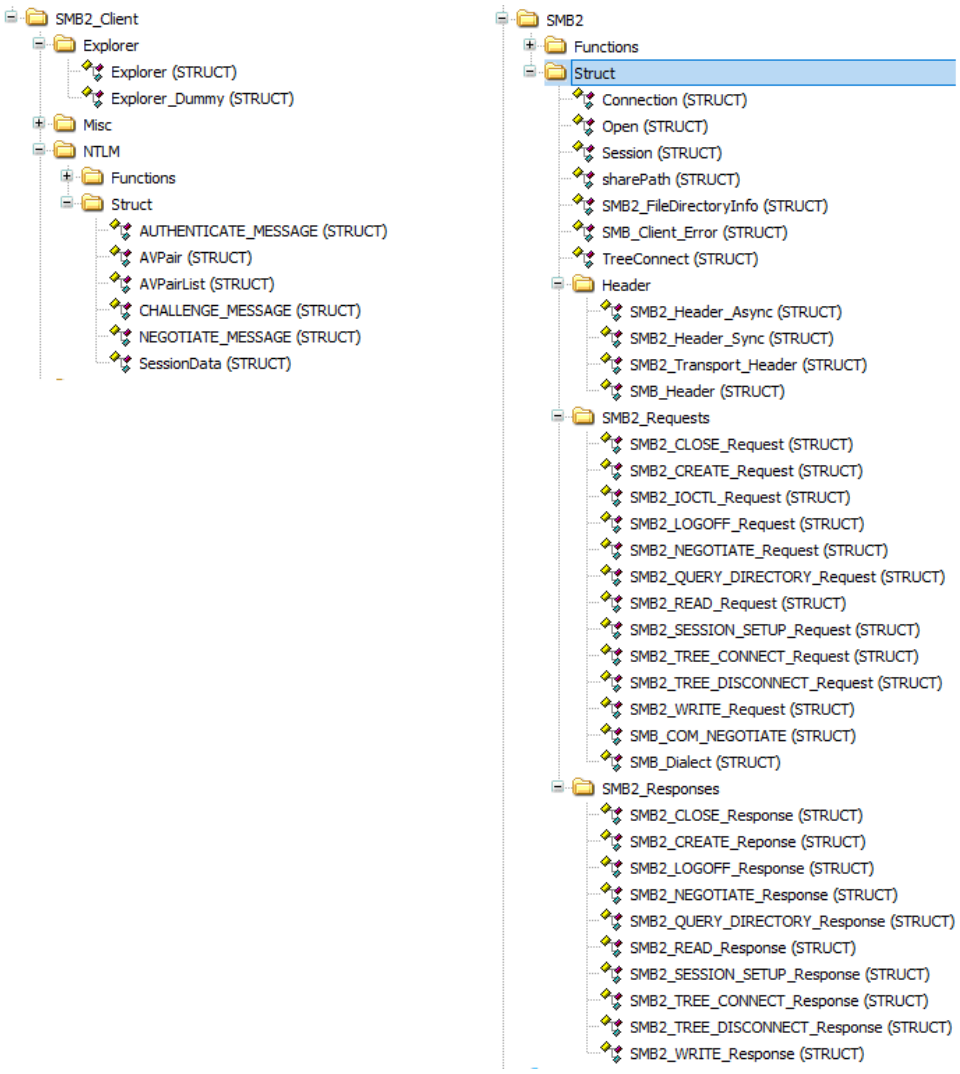


Abbildung 3.3: Strukturgruppierung

In den folgenden Unterkapiteln wird näher auf die implementierte Software eingegangen. Dazu werden Datenstrukturen und Funktionen näher erläutert und auf der Aufbau der Software mittels Programmablaufplänen sowieso UML Charts erklärt. Um die komplette Software detailliert zu betrachten, kann das kommentierte Programm im Anhang eingesehen werden.

3.3 Server Message Block Protocol

Die Entwicklung des SMB2 Client unterteilt sich in zwei Bereiche, zum einen wäre da die Dateiübertragung und zum anderen die Explorer Funktion. Auf beide wird separat eingegangen.

Implementiert wurde der „SMB2_Client“-Baustein. Ein späterer Anwender, welcher die Übertragung nutzen will brauch nur diesen Baustein aufrufen. Alle anderen implementierten Bausteine werden darin benutzt und sind für den Anwender nicht relevant. In Abb. 3.4 ist der Baustein dargestellt. Es folgt einen Beschreibung der Schnittstelle.

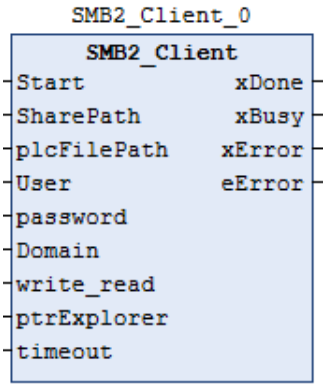


Abbildung 3.4: Schnittstellen des SMB2 Clients

INPUT

- Start : BOOL (positive Flanke startet die automatische Übertragung)
- SharePath : WSTRING (Dateipfad zum Server, siehe Abb. 3.5)
- plcFilePath : STRING (Dateipfad zur Datei auf der SPS)
- User : WSTRING (Benutzername zum einloggen - Bei Verbindung ohne Kennwort "Gast" wählen)

- Password : WSTRING (Passwort - Bei Verbindung ohne Kennwort leerer WSTRING)
- Domain : WSTRING (Domain der SPS z.b. "CoDeSys")
- write_read : INT (1 = SPS -> PC , 2 = PC -> SPS)
- ptrExplorer : Explorer (Pointer zur Explorer Variable, nur benötigt falls Explorer Mode genutzt wird)
- timeout : INT (Timeout Zeit in ms)

OUTPUT

- xDone : BOOL (Transfer ohne Fehler beendet)
- xBusy : BOOL (Transfer ist aktiv)
- xError : BOOL (Fehler bei der Übertragung)
- eError: SMB_Client_Error : (Fehlercode)

Der Dateipfad zum Server wird wie in Windows üblich angegeben. Dies ist in Abb. 3.5 zu sehen. Der erste Teil ist der Servername oder die IP-Adresse des Servers, mit dem sich verbunden werden soll. Der zweite Teil kennzeichnet den Namen des freigegeben Ordners, auch Share Name genannt. Der Rest gehört dabei zum Dateinamen, dies beinhaltet auch vorangestellte Ordnernamen.

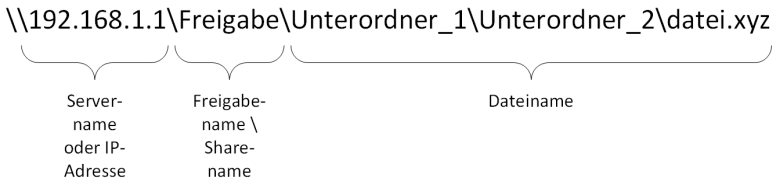


Abbildung 3.5: Beispiel: Pfad zu einer Datei

Die Ausgangsvariable eError setzt sich aus verschiedenen Variablen zusammen.

- Error_1 : INT (Kennzeichnet den Schritt in welchem der Fehler aufgetreten ist)
- Error_2 : DWORD (Beinhaltet den empfangenen SMB2 Fehlercode)
- TCP : NBS.ERROR (Beinhaltet den Fehler der TCP Verbindung)
- ConRST : BOOL (Kennzeichnet ob ein Connection-Reset vom Server veranlasst wurde)

3.3.1 TCP Funktionen

Die Funktionen für den Verbindungsaufbau, Senden und Empfangen sind CoDeSys eigenen Bausteine aus der Bibliothek Network. Diese sind sehr einfach zu benutzen und kümmern sich um alles auf der Transport Ebene. Die Funktionen selbst werden zyklisch in SMB2_Client aufgerufen. Über deren Eingangsvariablen werden sie gesteuert. Auf die Funktion zum empfangen wird etwas näher eingegangen, da die Benutzung nicht ganz trivial ist.

TCP_Client
Der TCP_Client baut die Verbindung zum Server auf, er benötigt nur die IP-Adresse oder den Servernamen. Nach erfolgreichem Verbindungsaufbau gibt er einen gültigen connection handle aus. Solange die Verbindung steht ist xActive = True.

TCP_Write
Mit dieser Funktion werden Daten zum Server gesendet. Zuerst schreibt man alle Daten in einen Buffer bzw. ein Byte Array. Anschließend wird ein Pointer zu diesem Array an die Funktion übergeben, zusätzlich wird die Länge der zu sendenden Daten benötigt. Nach erfolgreicher Übertragung meldet er xDone = True.

TCP_Read
Prinzipiell funktioniert die TCP_Read Funktion ähnlich wie TCP_Write. Man gibt einen Buffer und eine Buffergröße an und er wird zyklisch aufgerufen. Immer wenn neue Daten eingetroffen sind meldet der Baustein xReady = True. Aufgrund der TCP Architektur werden Daten aber in Paketen mit maximal 1460 Byte Nutzdaten verschickt. Immer wenn der Baustein aufgerufen wird und schon wieder ein neues Paket eingetroffen ist meldet der Baustein dies. Aufgrund dessen musste etwas entwickelt werden um größere Daten, aus mehrere Paketen, wieder zu einem zusammen setzen zu können.

Zuerst wird hierfür die Funktion weiterhin zyklisch aufgerufen. Da wir wissen, dass die ersten Daten immer den Header enthalten, extrahieren wir hieraus die Länge der gesamten Nachricht. Anschließend können wir dies mit der Länge der empfangenen Daten vergleichen. Stimmt diese überein wurde alles empfangen. Sind die empfangenen Daten kürzer als die erwarteten muss der Baustein die nächsten Daten hinten anhängen. Dazu wird dem Baustein im nächsten Zyklus eine neue Bufferposition übergeben und in einer Variable die Länge der empfangen Daten aufaddiert. Nach einer endlichen Anzahl von Durchläufe stimmt die erwartete Länge mit der aufaddierten Länge überein und die Daten im Buffer können weiterverarbeitet werden. In Abb. 3.6 ist der Ablauf zu sehen.

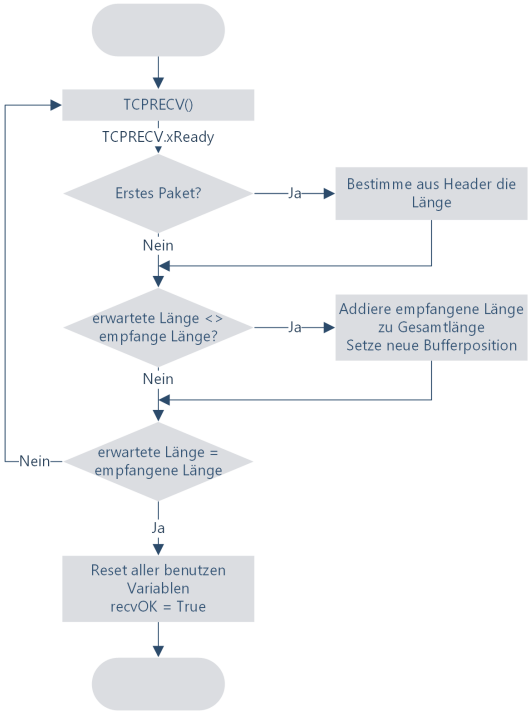


Abbildung 3.6: Ablauf Empfang von Daten

Auswertung der Empfangenen Daten

Immer wenn mittels TCP_Read alle Pakete empfangen, und das Bit recvOK gesetzt wurde, können die Daten ausgewertet werden. Auch dies passiert direkt im SMB2_Client Baustein. Es wird zyklisch überprüft ob das Bit gesetzt wurde, wenn ja wird der Header aus dem Buffer in eine Struktur kopiert und das Kommando, eine Zahl, ausgewertet. Mit diesem wird entschieden in welche Struktur die restlichen Daten kopiert werden. Danach wird das recvOK Bit wieder zurück gesetzt und die Daten in der Struktur werden im Programmablauf weiter verarbeitet.

3.3.2 Dateiübertragung

Die Dateiübertragung überträgt automatisch Dateien bzw. liest diese von einem Server. Die Quelle bzw. das Ziel auf diesem ist ein freigegebener Ordner. Die Daten auf der SPS werden auf der SD-Karte gespeichert. In diesem Unterkapitel gehe ich auf alle für die Dateiübertragung relevanten Bausteine und Datenstrukturen ein. Die Authentifizierungsbausteine werden in einem separatem Kapitel behandelt.

Datenstrukturen

Auf die Datenstrukturen in den Gruppen Request und Response wird nicht weiter eingegangen, diese können dem Dokument MS-SMB2[5] entnommen werden. Diese wurden genau wie beschrieben in eine Datenstruktur übernommen. Wichtig ist dabei nur der Parameter {attribute 'pack_mode' := '1'} mit welcher die Struktur angelegt wurde. Dieser gibt an, dass der Compiler die Daten am Byte ausgerichtet anlegt. So werden Lücken im Speicher verhindert und das kopieren der Daten kann Byteweise erfolgen. Selbst erdachte Strukturen sind folgende.

Connection
: Diese Struktur beinhaltet alle wichtigen Daten und Parameter zur aktuell aufgebauten Verbindung. Zum einen Daten wie Benutzername und Passwort aber auch vom Server geforderten Parameter wie z.b. die maximale Transaktionsgröße.

Session
: Session ist ein Teil der Connectionstruktur und beinhaltet alle wichtigen Parameter zur aktuellen Session, welche mit dem Session Setup etabliert wurde. Z.b. die SessionID und den Session Key.

TreeConnect
: Ebenfalls teil von Connection beinhaltet dies alle Parameter zur aufgebauten Verbindung zu einem Tree. Dies sind der Freigabename und die zugehörige TreeID mit welcher verbunden ist.

Open
: Open beinhaltet die FileId der geöffneten Datei, die Dateigröße, die Länge der gelesenen Daten und den aktuellen Offset in der Datei.

sharePath
: Diese Struktur dient dazu den Servernamen, die Freigabe Namen und den Dateinamen separat zur späteren Verwendung abzulegen.

Funktionen und Funktionsbausteine

Gemäß dem Sequenzdiagramm aus den Grundlagen wurde der Baustein für die Dateiübertragung erstellt. In Abb. 3.7 ist der gesamte Ablauf mit aufgerufenen Funktionen dargestellt. Die Verzweigung zum Explorer Mode wird in Kap. 3.3.3 näher beschrieben. Im Anhang sind Wireshark Übertragungen aufgezeichnet worden um den Verlauf einer Übertragung besser zu verstehen.

Auf die benutzten Funktionen wird im folgenden eingegangen.

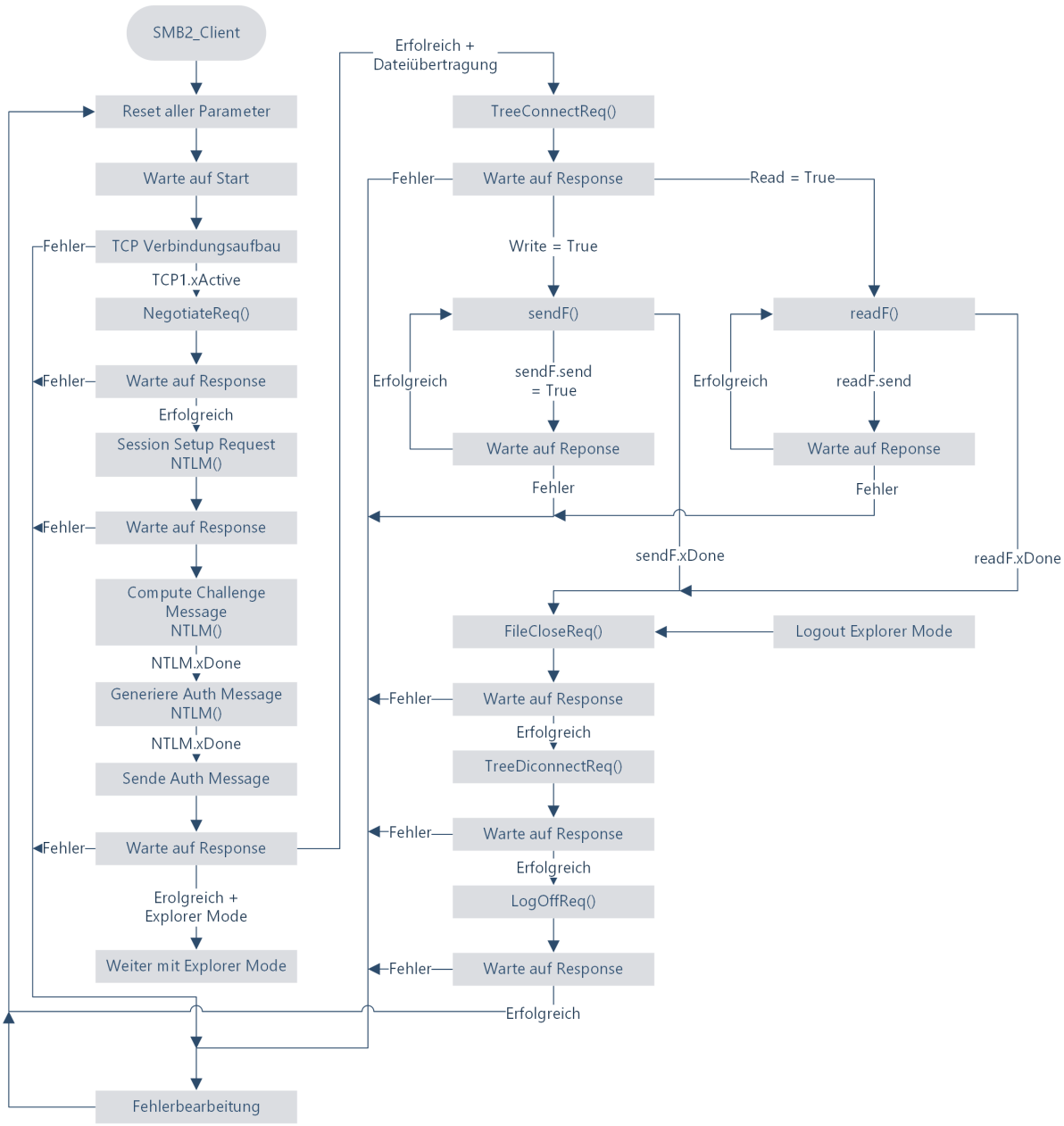


Abbildung 3.7: Ablauf der Dateiübertragung

Die implementierten Bausteine kommunizieren fast alle über die Variable „ConnectionParam“, welche die Datenstruktur „Connection“ abbildet.

decodeSharePath
Diese Funktion ermittelt aus dem am SMB2_Client angegeben SharePath die einzelnen Komponenten. Das heißt die Funktion zerlegt den String in IP-Adresse oder Servername, Freigabename und Dateiname. Die neuen Strings werden über eine Ausgangsvariable in die sharePath Struktur übergeben.

NegotiateReq
Dieser Baustein erstellt den SMB2 Header und den Negotiate Request. Es werden die entsprechenden Parameter und die Protokollversion übergeben und in den Sendebuffer kopiert.

TreeConnectReq
Der Baustein erhält als Übergabeparameter den Freigabe Namen des Ordners, mit dem verbunden werden soll. Daraufhin stellt die Funktion alle benötigten Parameter zusammen und übergibt sie dem Sendebuffer.

CreateReq
Diese Funktion sendet einen Request mit Dateinamen, zum öffnen einer Datei, an den Server. Je nach gewünschten Modi wird die Datei geöffnet, überschrieben oder neu erstellt. Anschließend sendet der Server die FileID der entsprechenden Datei zurück.

FileCloseReq
Der Baustein stellt einen File Close Request zusammen und übergibt sie dem Sendebuffer. Es wird sich immer von der einzig geöffnet Datei getrennt, die FileID bezieht der Baustein aus der übergebenen ConnectionParam Variable

TreeDisconnectReq
Die Funktion stellt alle relevanten Daten für den Tree Disconnect zusammen und übergibt sie dem Sendebuffer. Es wird sich immer vom aktuellen Tree getrennt, die TreeID bezieht der Baustein aus der übergebenen ConnectionParam Variable.

LogOffReq
Die Funktion stellt alle relevanten Daten für den LogOff der Session zusammen, damit wird sich komplett vom Server abgemeldet.

SendFile
Der Funktionsbaustein sendF bzw. sendFile übernimmt das lesen von SD-Karte und das Senden einer Datei an den Server. Er wird im Verlauf der Kommunikation aufgerufen und bearbeitet. Mit einen xDone meldet dieser die erfolgreiche Übertragung der Datei. Siehe Abb. 3.8 für den gesamten Ablauf.

Die drei Funktionen getSize, open, read und close sind Systemfunktionen, welche auf die SD-Karte der SPS zugreifen. Zuerst wird mit der Funktion getSize die Größe der zu übertragenden Datei bestimmt und mit open die Datei geöffnet. Aus dem daraus ermittelten Wert und der Buffergröße wird berechnet wie viele einzelne Write Request gesendet werden müssen. Nach dem erfolgreichen Create Request wird der Write Request zusammen gestellt und mittels read die Daten von der SD-Karte in den Sendebuffer kopiert. Dies wiederholt sich solange bis die ganze Datei übertragen wurde. Anschließend wird die Datei mit close wieder geschlossen und der Output xDone gesetzt.

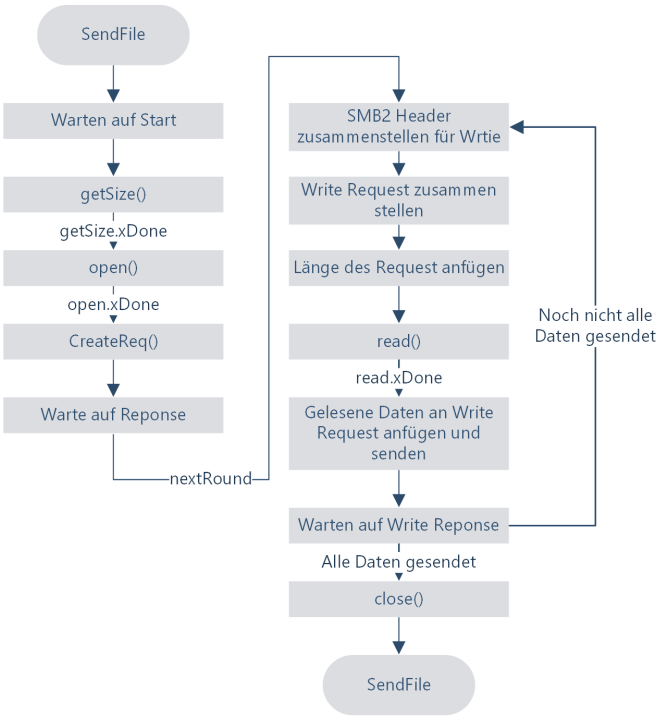


Abbildung 3.8: Ablauf des SendFile Bausteins

ReadFile

Der Funktionsbaustein readF bzw. ReadFile übernimmt das empfangen einer Datei und das schreiben auf die SD-Karte. Er wird im Verlauf der Kommunikation aufgerufen und bearbeitet. Mit einem xDone meldet dieser die erfolgreiche Übertragung der Datei. Siehe Abb. 3.9 für den gesamten Ablauf.

Wie schon bei Sendfile werden die Systemfunktionen für die SD-Karte benutzt. Am Anfang wird ein Create Request mit Dateiname und den Optionen zum öffnen der entsprechenden Datei gesendet. Falls diese auf dem Server vorhanden ist erfolgt der

Response positiv. Anschließend wird eine neue Datei, mit open, auf der SD-Karte angelegt oder eine vorhanden überschrieben. Im Create Response wurde die Größe der Datei empfangen, diese Daten nutzen wir um zu berechnen wie oft gelesen werden muss. Dies hängt von der Größe der Datei und der Buffergröße ab. Danach wird der Read Request erstellt und die Daten im Read Response empfangen. Diese Daten werden mittels write auf die SD-Karte geschrieben. Solange nicht alle Daten empfangen wurden muss dieser Vorgang wiederholt werden. Danach wird die Datei mit close geschlossen und der Baustein meldet xDone fertig.

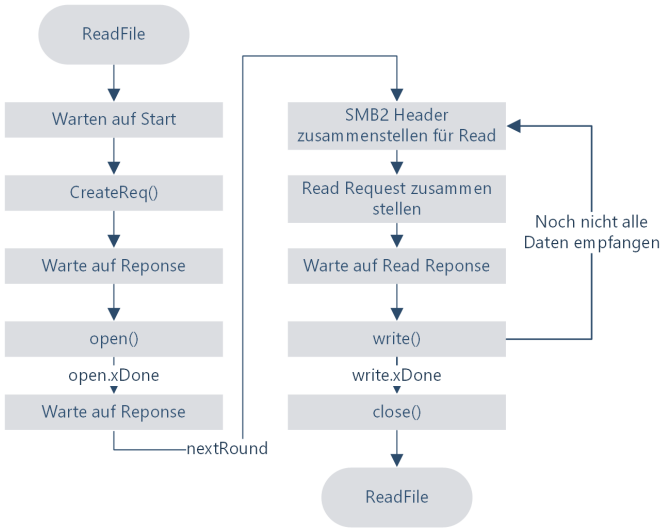


Abbildung 3.9: Ablauf des ReadFile Bausteins

getSize, open, write, read, close

Diese fünf Funktionen sind für das lesen und schreiben von Daten auf SD-Karte zuständig. Der interne Aufbau ist unbekannt, da es vorkompilierte Funktionen des Herstellers sind. Vor einem read oder write muss die Datei mit Open geöffnet werden. Open erzeugt einen File Handle, mit welchem read/write auf die geöffnete Datei zugreift. Mit close muss die Datei wieder geschlossen werden. Die Funktion getSize benötigt keinen File Handle und kann jederzeit aufgerufen werden um die Größe einer Datei auf SD-Karte zu ermitteln.

3.3.3 Explorer

Mittels Explorer Mode wird in der WebVisu der SPS eine Art Dateiexplorer implementiert. Diese wird im Webbrowser geöffnet, Adresse „http://ip-adresse:8080/webvisu.htm“. Über

diese Visualisierung können die Verbindungsdaten wie IP-Adresse, Benutzername, Passwort und Domain, des Servers, eingeben und sich eingeloggt werden. Beim ersten einloggen ermittelt der SMB2_Client die Namen der freigegeben Ordner. Anschließend können diese geöffnet und die Namen von Ordnern und Dateien darin aufgelistet werden. Zusätzlich kann mit einem Klick auf „übernehmen“ der komplette Pfad zur automatischen Übertragung kopiert werden. Oder man bearbeitet diesen von Hand. Der Pfad auf der SPS muss von Hand eingegeben werden, ein durchsuchen ist nicht möglich. Nach ausloggen ist der SMB2 Client Baustein wieder bereit für automatische Übertragung, diese kann auch von Hand in der WebVisu gestartet werden.

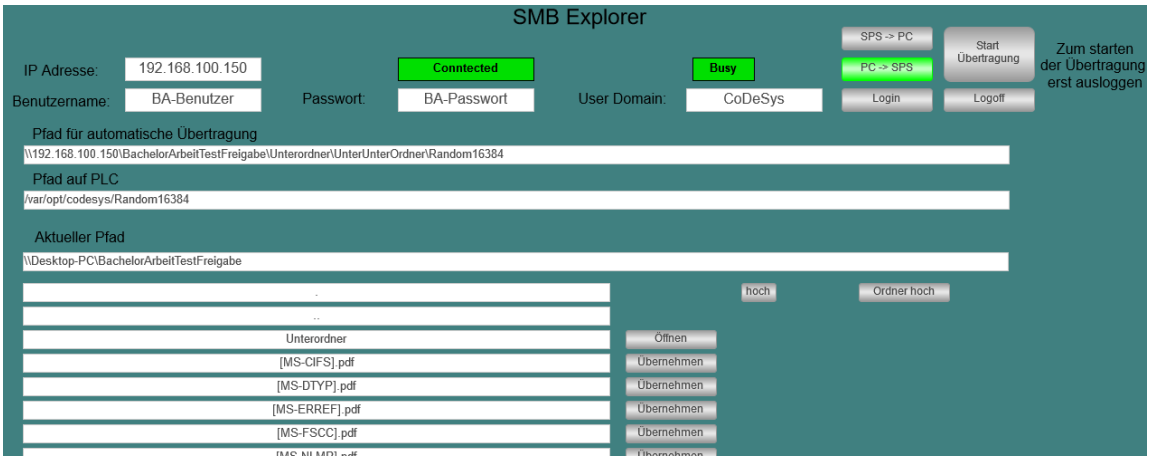


Abbildung 3.10: Explorerfenster in der WebVisu

Auf Abb. 3.10 ist der Explorer zu sehen um alle Funktionen nutzen zu können muss der Baustein wie in Abb. 3.11 konfiguriert sein.

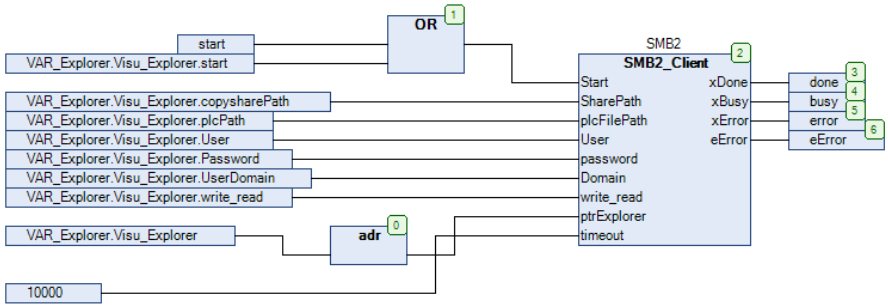


Abbildung 3.11: Konfiguration Explorermode

In diesem Unterkapitel gehe ich auf alle für den Explorer Mode relevanten Datenstrukturen und Bausteine ein.

Datenstrukturen

Auf die Strukturen der Requests und Responses gehe ich auch hier nicht weiter ein. Diese sind wie bei der Dateiübertragung in der Microsoft Spezifikation zu finden. Größtenteils werden auch schon vorhandene Datenstrukturen weiter verwendet.

Explorer
: Die Explorer Datenstruktur beinhaltet die Konfiguration und alle Variablen mit denen zwischen WebVisu und SMB2_Client-Baustein kommuniziert wird. Außerdem beinhaltet es einen Zwischenspeicher für ausgelesene Ordner und Dateinamen.

SMB2_FileDirectoryInfo
: Diese Datenstruktur kommt beim Auslesen der Ordner- und Dateinamen in einem Verzeichnis zum Einsatz. Es bildet die Informationen über einen solchen Namen mit detaillierten Attributen, siehe MS-FSCC[6] Kap. 2.4.10. Dies wird beim Empfang der Liste zum herausfiltern der Namen und Unterscheidung zwischen Ordner oder Datei genutzt.

Funktionen und Funktionsbausteine

Einige Funktionen, welche wie bei der Dateiübertragung verwendet werden, werden hier nicht weiter behandelt. Falls Funktionen wieder verwendet aber etwas spezieller aufgerufen werden gehe ich erneut darauf ein. Das herausfiltern der Freigabenamen und Ordner-/Dateinamen wird gesondert erklärt.

In Abb. 3.12 ist der Explorer Mode abgebildet. Da ein- und ausloggen den allgemeinen Ablauf wie bei der Dateiübertragung nutzt fängt die Abbildung auch erst an den relevanten Stellen an. Nach der erfolgreichen Anmeldung werden zuerst die Freigabenamen ermittelt, nach einem Klick auf Öffnen, in der WebVisu, die darin enthalten Ordner- und Dateinamen.

CreateReq
: Die Funktion ist die gleiche wie bei der Dateiübertragung. Für das herausfinden der Freigabenamen muss allerdings der Name „srvsvc“ übergeben werden. Beim herausfinden der Ordner- und Dateinamen wird ein leerer String übergeben.

NetSharebind
: Diese Funktion stellt einen Write Request zusammen. Die Daten welche gesendet werden müssen wurden aus einer bestehenden Verbindung mittels Wireshark ermittelt. Es wurde dazu keine Dokumentation bei Microsoft gefunden. Der Bind muss geschehen um Server Funktionen nutzen zu können.

NetShareEnumAll

Diese Funktion stellt einn Ioctl (input/output control) Request zusammen. Mit den angehängten Daten können auf dem Server Befehle aufgeführt werden. In unserem Fall wird gesagt, dass wir gerne die Liste mit Freigabenamen hätten. Auch hier stammen die Daten, die gesendet werden, größtenteils aus Wireshark.

QueryDirReq

Es wird eine Query zusammen gestellt. Im Prinzip ist dies eine Suchfunktion auf dem Server. Bevor die Funktion genutzt werden kann, wird der CreateReq mit leerem Dateinamen benötigt. Nach dem Request bekommt man eine Liste mit Ordner- und Dateinamen und den zugehörigen Dateiattributen. Man sendet im Request das Format, welches man zurück bekommen möchte. Zusätzlich einen String nach dem gesucht werden soll. Da alles ausgelesen wird besteht dieser aus „*“. Der String ist fest vergeben, es kann mit der Funktion ohne Veränderung nach nichts anderem gesucht werden.

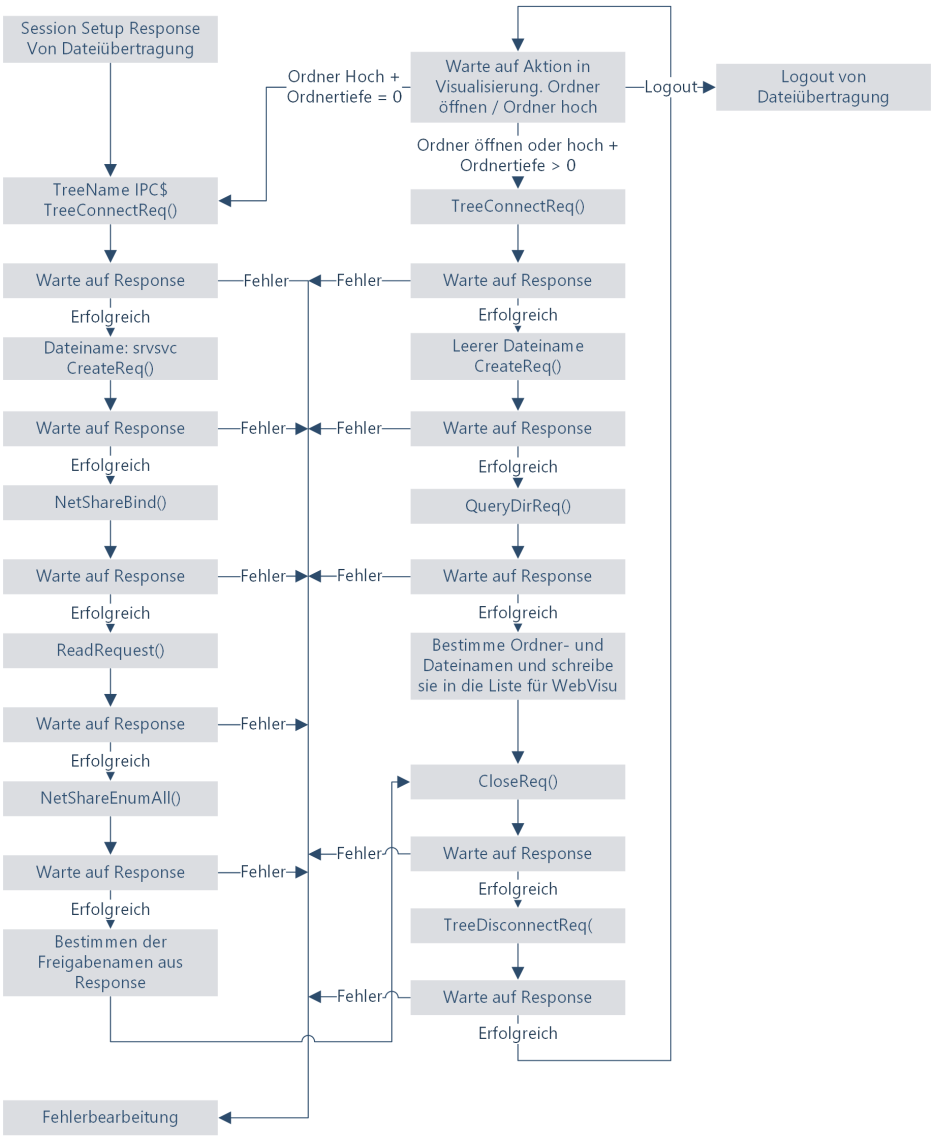


Abbildung 3.12: Ablauf des Explorer Mode

Herausfiltern der Freigabenamen

In Abb. 3.13 ist der Ablauf zum herausfiltern der Freigabenamen dargestellt. Dies ist nicht ganz trivial, da keine Dokumentation zum Aufbau der Daten gefunden werden konnte. Zusätzlich war auch Wireshark nicht in der Lage, die Daten korrekt darzustellen. Beim betrachten im Hex-Editor konnte der Aufbau von Dateinamen und deren Länge erkannt werden.

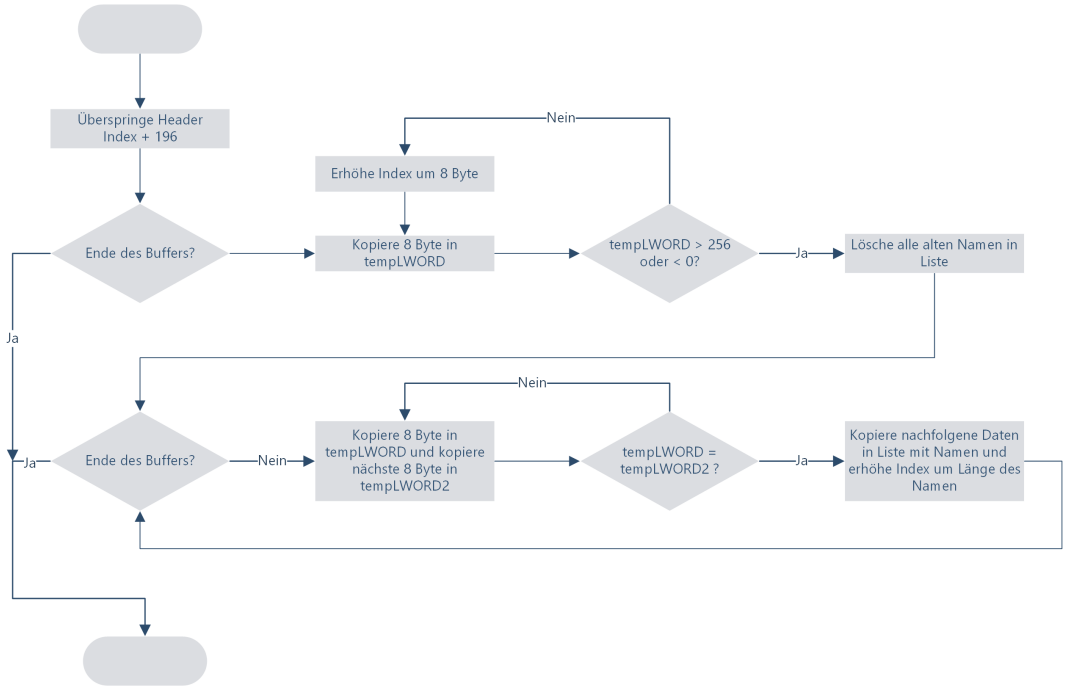


Abbildung 3.13: Ablauf Freigabenamen herausfiltern

Nach NetShareEnumAll bekommt man einen Response mit einer angehängten Liste. Diese Daten stehen nun im Buffer. Zuerst wird der Header übersprungen. Anschließend wird nach dem Anfang der Namen gesucht. Der Aufbau der Namensliste ist immer gleich, die ersten 8 Byte enthalten die Länge, dann folgen 8 Byte Nullen, darauf wieder die Länge. Daraufhin der Name inklusive nachfolgende Paddingbytes um auf 8 Byte Blöcke zu kommen. Der erste Name ist immer ein Freigabename, der darauffolgende ein Kommentar.

Die beiden Längenangaben werden verglichen und falls gleich der Name in ein Stringarray kopiert. Der Kommentar wird zwar ausgewertet aber verworfen.

Herausfiltern der Datei- und Ordnernamen

In Abb. 3.14 ist der Ablauf zum herausfiltern der Ordner- und Dateinamen dargestellt.

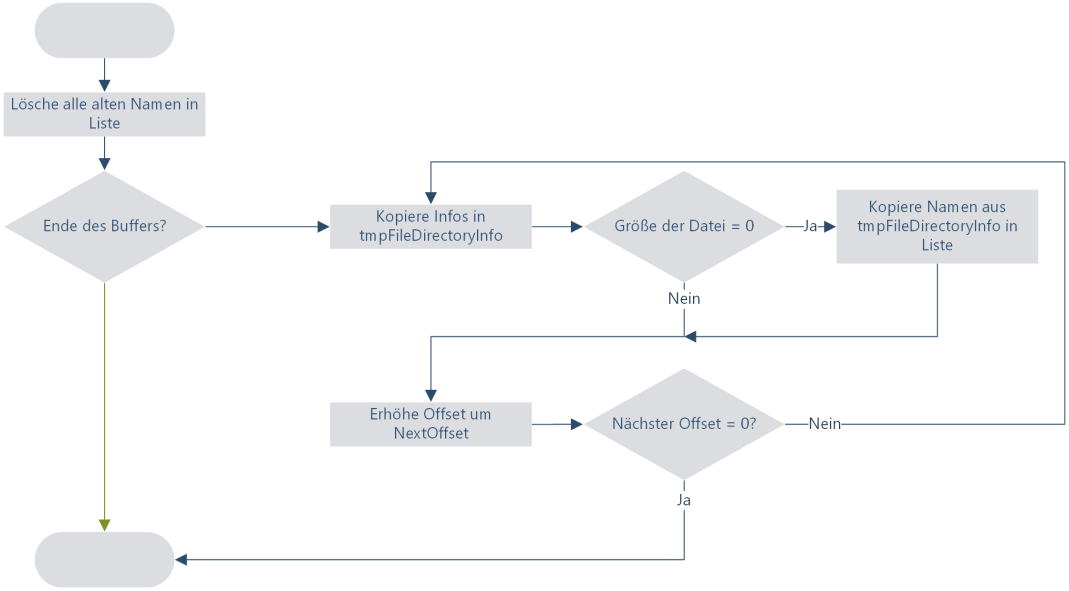


Abbildung 3.14: Ablauf Ordner- und Dateinamen herausfiltern

Mit dem Query Directory Response bekommt man eine Liste mit allen Infos. Je nach angefragten Details kann diese Info anders aussehen. In der Software wird aber die komplette Info benötigt, da diese neben Uhrzeiten der Erstellung auch die Größe und Attribute einer Datei beinhaltet. Anhand dieser Attribute kann zwischen Ordner und Datei unterschiedenen werden. In Abb. 3.14 ist nur der Ablauf für die Ordnernamen dargestellt. Ordner sind in der Liste immer mit dem Attribut „Directory“ markiert. Für Dateinamen wurde der ganze Ablauf erneut verwendet und anstatt auf Directory wurde geprüft of das Attribut nicht vorhanden ist.

Warten auf Eingabe in der Visualisierung

Nach dem Einloggen in der Visualisierung wird sich mittels SMB2 am PC eingeloggt und direkt die Freigabenamen abgerufen. Danach wird auf Eingabe in der Visualisierung gewartet. Dies geschieht in Schritt 500 im SMB2_Client-Baustein.

Mit einer Variable wird die Ordnertiefe gespeichert. Wird ein Ordner geöffnet wird diese erhöht und die Namen darin abgerufen. Wird der Button „Ordner hoch“ angeklickt wird die

Variable um eins verringert und wiederum die Namen darin abgerufen. Wenn die Variable allerdings null ist werden wieder die Freigabenamen abgerufen. Zu sehen in Abb. 3.12.

3.4 NT LAN Manager Authentication Protocol

Die Authentifizierung passiert während des Session Setups. Die Kommunikation mit dem Server geschieht hier auch nur als Teil des Session Setups, alle Daten werden diesem an gehangen.

Die Authentifizierung geschieht ist 3 Schritten, zuerst wird dem Server eine Negotiate Nachricht übersendet, auf diese antwortet er mit einer Challenge Message, mit Hilfe des Benutzernamens und Passwort werden daraus die Antworten berechnet und die Authentication Message versandt.

Wie diese Daten zu berechnen sind wird ausführlichem mit Hilfe von Pseudocode, im MS-NLMP[4] von Microsoft beschrieben.

Hash- und Verschlüsselungsfunktionen

Die benötigten Funktionen wurde übernommen oder nach Vorgabe implementiert.

MD4
: Diese Funktion wird in der Software nur einmal zum Hashen den Passworts benötigt, aufgrund dessen wurde sie nicht für zyklische Bearbeitung implementiert. Der Code stammt aus RFC1320 [8].

MD5
: MD5 ist eine fertige Funktion und wurde aus der CoDeSys Network Bibliothek entnommen. Sie wird zyklisch aufgerufen bis der Hashwert feststeht.

HMAC_MD5
: HMAC wurde mit Hilfe der fertigen MD5 Funktion implementiert. Daher wurde auf die zyklische Abarbeitung geachtet. Die Grundlage zur Implementierung stammt aus RFC2104 [3].

RC4
: RC4 wurde mit verschiedenen Modes implementiert. Zu einen kann es nur mit einem Schlüssel initialisiert, die Nachricht verschlüsselt/entschlüsselt oder beides ausgeführt werden.

3.4.1 Negotiate Message

Der Negotiate wird im ersten SMB2 Session Setup übermittelt. Um es zusammen zu stellen wird die Funktion NTLM mit Mode = 1 aufgerufen. Der Anhang vom Session Setup besteht aus einem Object Idetifier (OID), welcher immer gleich ist, einer MechTypeList, in dieser stehen die gewünschten Authentifizierungsmethode, hier nur ein Eintrag für die implementierte Methode, und der Negotiate Message.

Im ersten Aufruf (Mode = 1) werden diese Object Idetifier, MechTypes und die Negotiate Message zusammen gestellt. Die Nachricht selbst besteht aus den gewünschten Flags. Domain und Workstation Namen des Clients sind leer.

In Abb. 3.15 ist der Aufbau der Nachricht in Wireshark zu sehen.

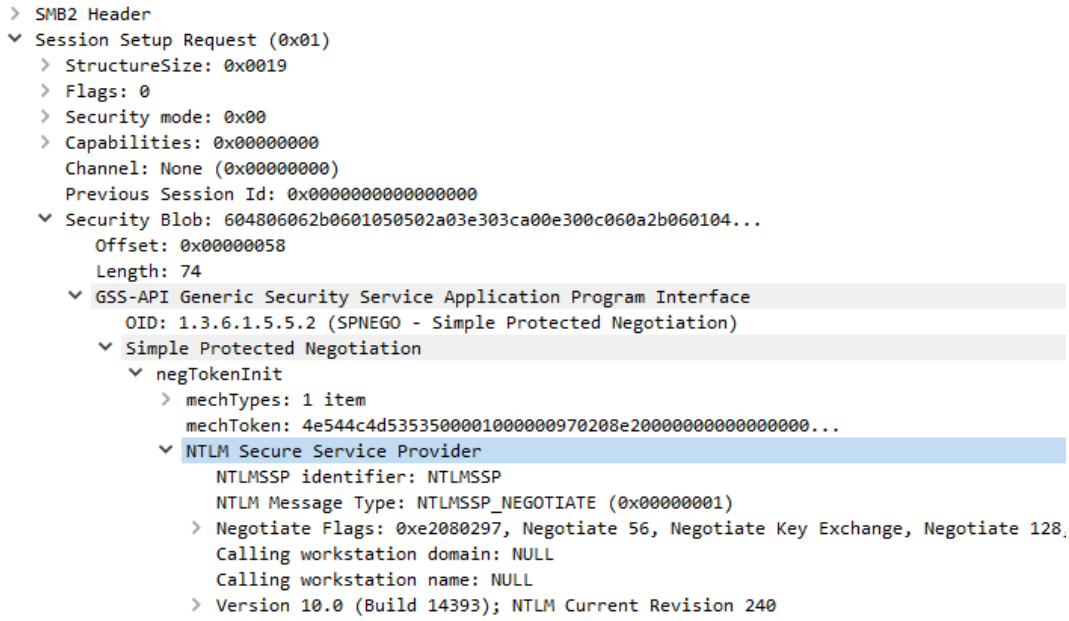


Abbildung 3.15: Session Setup Request und Negotiate Message

3.4.2 Challenge Message

Im Session Setup Response übermittelt der Server eine Challenge Message. Diese, bzw. der Empfangsbuffer, wird der NTLM Funktion mit Mode = 2 übergeben. Zuerst wird im Buffer nach dem Anfang der Nachricht (NTLMSSP) gesucht und diese in eine Struktur kopiert um damit weiter arbeiten zu können.

Der Aufbau, wie vom Server gesendet ist in Abb. 3.16 zu sehen. Darin zu erkennen, dass der Server bei supportedMech mit der gleichen Nummer, wie bei Negotiate gesendet, antwortet und anschließend die Challenge Message angehängt ist.

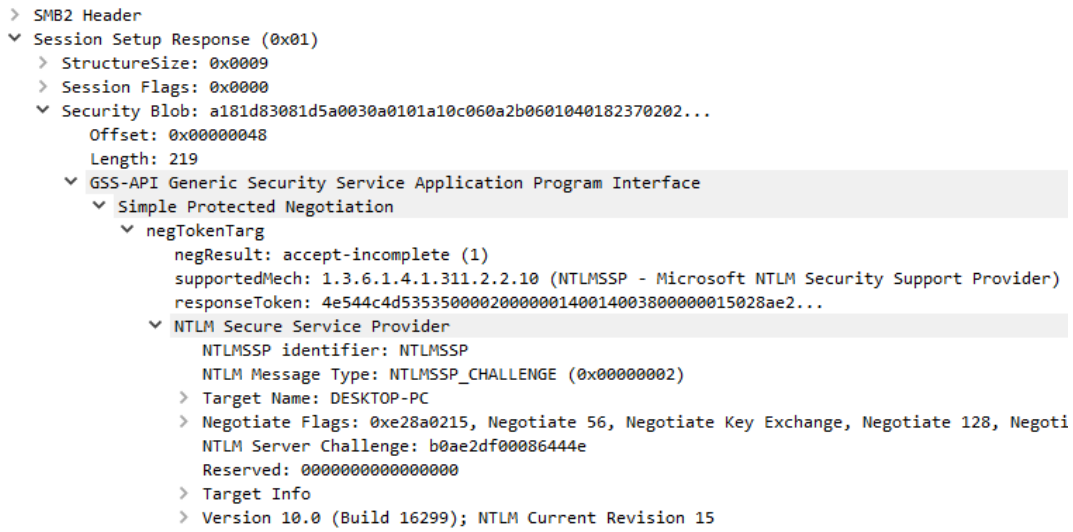


Abbildung 3.16: Session Setup Respone und Challenge Message

Funktionen und Funktionsbausteine

GetAVPairList

Diese Funktion ermittelt als erstes die Attribute, welche uns in der Challenge Message übersandt wurden. Die Attribute sind immer unterschiedlich, meinst bestehen sie aus dem Servernamen, Domainname und Zeitstempel. Diese werden einzeln extrahiert und für spätere Verwendung gespeichert.

ProcessChallengeMessage

In dieser Funktion wird die Challenge Message verarbeitet, es werden weitere Funktionen aufgerufen. Der Ablauf wurde aus MS-NLMP [4] Kap 3.1.5.1.2 und Kap 3.3.2 übernommen und auf zyklische Abarbeitung angepasst.

NTOWFv2 und LMOWFv2

Diese Funktion berechnet aus Benutzername, Passwort und Domain den ResponseKeyNT bzw. ResponseKeyLM. Es werden die Funktionen MD4 und HMAC_MD5 benötigt. Der Code wurde aus MS-NLMP [4] Kap 3.3.2 übernommen und auf zyklische Bearbeitung angepasst.

ComputeResponse
Diese Funktion berechnet aus der Server Challenge den NTProofStr, NtChallenge-Response, LMChallengeResponse und sessionBaseKey. Auch diese Funktion wurde komplett aus MS-NLMP [4] Kap 3.3.2 übernommen und auf zyklische Bearbeitung angepasst.

SetAVPairList
Diese Funktion erstellt aus Servernamen, Domainnamen und Zeitstempel eine neue Attributeliste. Diese wird als Byte Array gespeichert und in ComputeResponse weiterverarbeitet.

EncryptedSessionKey
Diese Funktion verschlüsselt den exportedSessionKey mittels RC4 und dem KeyExchangeKey als Schlüssel. Siehe MS-NLMP [4] Kap. 3.1.5.1.2.

SIGNKEY
Diese Funktion berechnet aus dem ExportedSessionKey und einem vom Microsoft vorgebenen String, mittels MD5, einen Client bzw. Server SigningKey. Mit diesem werden Nachrichten signiert. Die Funktion ist in MS-NLMP [4] Kap. 3.4.5.2 beschrieben.

SEALKEY
Ähnlich wie SIGNKEY berechnet diese Funktion den Client und Server Sealingkey. Wie lang dieser ist hängt allerdings von den Negotiate Flags ab. Siehe MS-NLMP [4] Kap. 3.4.5.3.

mechListMic
Diese Funktion berechnet aus der MechTypeList (gesendet bei Negotiate Message) und dem zurvor brechneten Client-SigningKey und Sealingkey eine 16 Byte lange Zahl zur Prüfung der Nachrichten Integrität. Diese Zahl wird hinter der Auth Message an gehangen.

3.4.3 Authenticate Message

Im zweiten Session Setup Request wird die Authenticate Message übermittelt, vorher wird die Funktion NTLM mit Mode = 3 aufgerufen. Die Funktion GenerateAuthenticateMessage generiert schon mit Mode = 2 alle nötigen Werte, diese werden nun benötigt. Sind alle Daten in die Auth Message kopiert wird diese gesendet. Ist alles korrekt sendet der Server einen Response mit einer gültigen SessionID und man kann sämtliche SMB2 Kommandos ausführen.

In Abb. 3.17 kann man die Auth Message mit ihren berechneten Werten sehen.

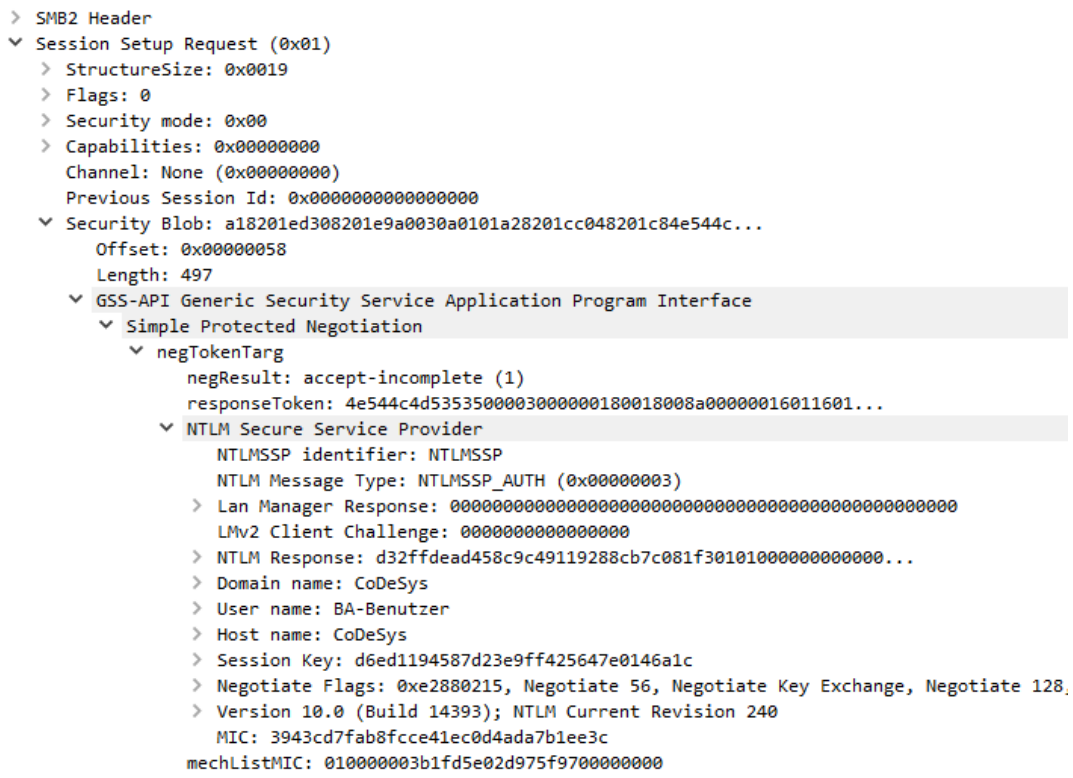


Abbildung 3.17: Session Setup Request und Authenticate Message

Funktionen und Funktionsbausteine

GenerateAuthenticateMessage

Diese Funktion stellt alle berechneten Werte, Namen und Flags zusammen und kopiert sie in die AuthMess Struktur. MIC wird dabei auf 0 gesetzt damit später der echte MIC (Message Integrity Check) berechnet werden kann.

MIC

Diese Funktion kopiert die Negotiate, Challenge und Auth Message hintereinander in einen Buffer. Anschließend wird mittels HMAC MD5 und dem exportedSessionKey der 16 Byte Wert berechnet.

3.5 Fehlerbehandlung

Die Fehlerbehandlung ist sehr einfach gehalten. Wenn die Dateiübertragung automatisch abläuft kann mit einem auftretenden Fehler nur die Übertragung abgebrochen werden.

Die Fehlerbearbeitung wird immer dann angesprungen wenn in einem Response das Statusfeld einen Fehler meldet. Diese sind meinst auf Konfigurationsprobleme beim Server zurückzuführen, z.b. falsche Benutzernamen/Passwort, fehlende Lese- und Schreibberechtigungen oder nicht existente Dateien.

Weitere Fehler die auftreten können sind beim lesen und schreiben der SD-Karte.

An jeder Stelle in der Software, an der Fehler passieren können, wird dies abgefragt. Falls einer Auftritt wird Schritt 900 angesprungen und Fehlercode, Schrittnummern usw. für den Benutzer ausgegeben. Anschließend wird die TCP Verbindung getrennt und der Baustein resettet.

4 Erprobung und Vergleich

In diesem Kapitel wird der Hardwareaufbau mit der entwickelten Software aufgebaut und auf Funktion geprüft, zusätzlich werden die Anforderungen validiert. Des weiteren wird die Performance der Dateiübertragung überprüft und verglichen.

4.1 Funktionstest

Für den Funktionstest wurde die Software kompiliert und übertragen. Die Dateien für das Senden an einen freigegeben Ordner wurde mittels in CoDeSys integriertem Dateiexplorer auf die SD-Karte des Raspberry Pi übertragen, siehe Abb. 4.1.

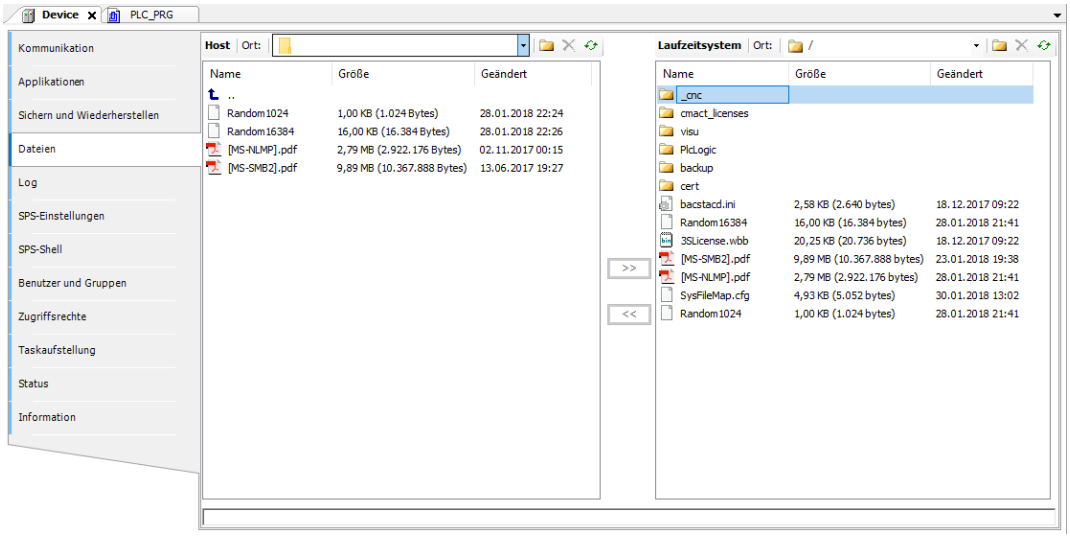


Abbildung 4.1: Dateiexplorer in CoDeSys Entwicklungsumgebung

Anschließend wurde der SMB2 Client Baustein wie in Abb. 4.2 konfiguriert. Für verschiedene Dateien wurde der Pfad entsprechend angepasst. Auf dem Windows PC wurde ein neuer Benutzer, mit dem Namen „BA-Benutzer“und Passwort „BA-Passwort“, angelegt. Der Benutzer bekam beim freigegeben Ordner Lese- und Schreibrechte.

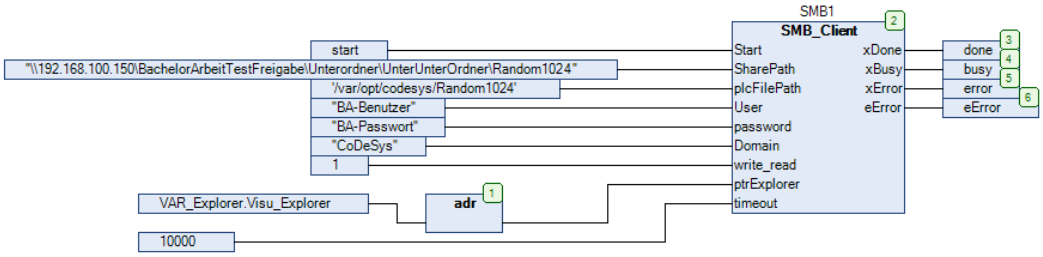


Abbildung 4.2: Konfiguration des Funktionstest

Anschließend wurde die Software mittels CoDeSys Entwicklungsumgebung auf die Steuerung übertragen. Zum testen der Funktion wurde auf dem Windowsrechner, welcher den SMB2 Server stellt, die Software „Wireshark “(www.wireshark.org) installiert. Damit können die gesendeten und empfangen Datenpakete aufgezeichnet und analysiert werden. So wurde die richtige Abfolge des SPS-Programms überprüft. Bei Fehlern konnten die entsprechenden Stellen leicht entdeckt und geändert werden.

4.1.1 Test der Dateiübertragung

Wie schon erwähnt wurden die Testdateien vorher auf die SD-Karte der SPS übertragen und der Baustein konfiguriert. Anschließend wurde die Übertragung mittels SMB2 gestartet und auf eine Rückmeldung des Bausteins gewartet. Zur Überprüfung wurden vier verschiedene Dateien verwendet. Zwei davon sind die Microsoft PDFs zum Protokoll, diese wurden gewählt weil sie relativ groß sind. Zusätzlich wurden zwei kleine Binärdateien auf der Seite random.org erzeugt. Eine Übersicht liefert Tab. 4.1. Die Dateien wurden jeweils mehrfach übertragen und der MD5 Hash der übertragenden Datei und Originaldatei verglichen.

Tabelle 4.1: Übersicht der Testdateien

Dateiname	Größe in Byte	MD5 Hash
[MS-SMB2].pdf	10.367.888	f44f905206b08f3d2792084516960e12
[MS-NLMP].pdf	2.922.176	53e5224499f9e0471d93ce169880fb35
Random16384	16384	7fa400fc9d1db0ef122907f896b3f5e1
Random1024	1024	56a9cfe844a858672c12ceeede1750a0

Es wurde zusätzlich verschiedene Buffergrößen getestet. Jeweils für beide Richtungen, SPS -> PC und PC -> SPS, wurden die Größen 8192, 16384 und 32768 Bytes getestet. Dies

wurde über eine Konstante im SMB2-Client Baustein eingestellt. Nach einigen Anpassungen wurden alle Dateien richtig übertragen, die Übertragung selbst läuft dabei ohne Probleme.

4.1.2 Test des Explorers

Die WebVisu wurde im Webbrowser aufgerufen, die IP-Adresse des SMB2 Servers und Benutzername/Kennwort eingegeben. Nach einem Klick auf Login und dem erfolgreichen verbinden wurde manuell überprüft ob die angezeigten Freigabenamen mit den freigegeben Ordner übereinstimmen. In den Unterordnern wurde überprüft ob alle darin enthaltenen weiteren Unterordner und Dateien korrekt angezeigt werden.

4.1.3 Performancetest

Beim Performancetest wurde die Übertragungsgeschwindigkeit für die vier Dateien ermittelt. Es wurden beim SMB2 Client verschiedene Buffergrößen getestet und mit einem fertigen FTP-Client verglichen. Viele Hersteller bieten für ihre Steuerungen einen FTP-Client an, entschieden wurde sich für den von OSCAT (www.oscat.de), dieser ist für verschiedene Hersteller nutzbar.

Die Zeiten wurden in Wireshark ermittelt. Dazu wurde die komplette Übertragung aufgezeichnet, der Verbindungsaufbau und die Trennung wurden als Zeitmarken genutzt und deren Differenz als Übertragungszeit gewertet. Jede Datei wurde dreimal übertragen und die Zeiten in Tabelle 4.2 notiert.

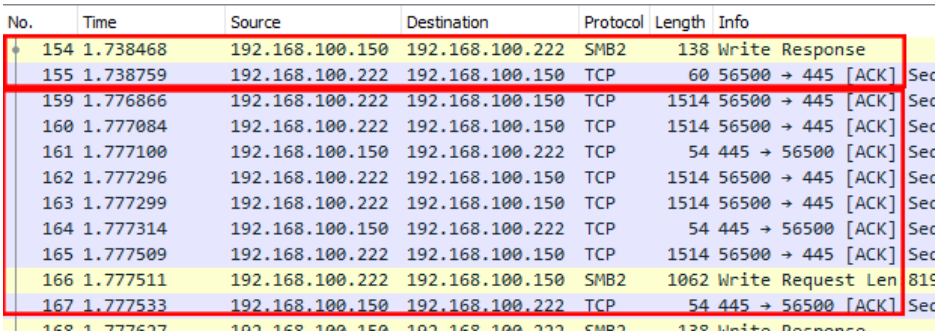
Tabelle 4.2: Ergebnisse Performancetest

		SPS -> PC			PC -> SPS		
	Buffer in Byte	Zeit 1 in *s*	Zeit 2 in *s*	Zeit 3 in *s*	Zeit 1 in *s*	Zeit 2 in *s*	Zeit 3 in *s*
[MS-SMB2].pdf - 10.367.888 Bytes							
FTP-Client		445,72	445,72	446,54	91,69	93,24	71,11
	8192	51,69	51,65	51,74	47,17	47,19	47,14
SMB2	16384	26,25	26,23	26,23	21,17	21,07	21,09
	32768	13,62	13,60	13,69	28,49	32,18	28,84
[MS-NLMP].pdf - 2.922.176 Bytes							
FTP-Client		125,71	126,12	126,11	20,21	21,13	20,20
	8192	15,18	15,15	15,12	13,90	13,85	13,85
SMB2	16384	8,05	8,01	8,00	6,59	6,53	6,63
	32768	4,52	4,47	4,46	9,36	9,20	9,34
Random16384 - 16384 Bytes							
FTP-Client		1,47	0,88	0,88	5,31	5,42	5,01
	8192	0,96	0,95	0,95	0,60	0,89	0,89
SMB2	16384	0.92	0.92	0.91	0,88	0,84	0,87
	32768	0,87	0,88	0,88	0,92	0,91	0,85
Random1024 - 1024 Bytes							
FTP-Client		0,90	0,92	0,91	5,24	5,38	5,15
	8192	0,92	0,91	0,90	0,82	0,82	0,83
SMB2	16384	0,92	0,88	0,88	0,89	0,84	0,83
	32768	0,88	0,88	0,87	0,83	0,83	0,82

Bei den größeren Dateien ist zu erkennen, dass der SMB2-Client gleich welcher Buffergröße, signifikant schneller ist als der FTP-Client. Bei kleineren Dateien ist kein großer Unterschied mehr zu bemerken. Auch die verschiedenen Buffer spielen keine große Rolle bei kleinen Dateien. Dies liegt daran, dass die Übertragung der Daten einen kleinen Teil in der gesamten Kommunikation einnehmen, während bei SMB2 dann der größte Teil von der Authentifizierung des Clients abhängt. Diese Zeit ist auch nicht weiter Reduzierbar. Auch zu erkennen ist, dass die Zeit bei PC -> SPS mit 32kb Buffer wieder zunimmt. Dieses Phänomenen rührt daher, dass es bei vielen TCP Segmenten, die richtig empfangen werden müssen, öfter zu Übertragungsfehlern und damit zu erneuten Übertragungen kommt.

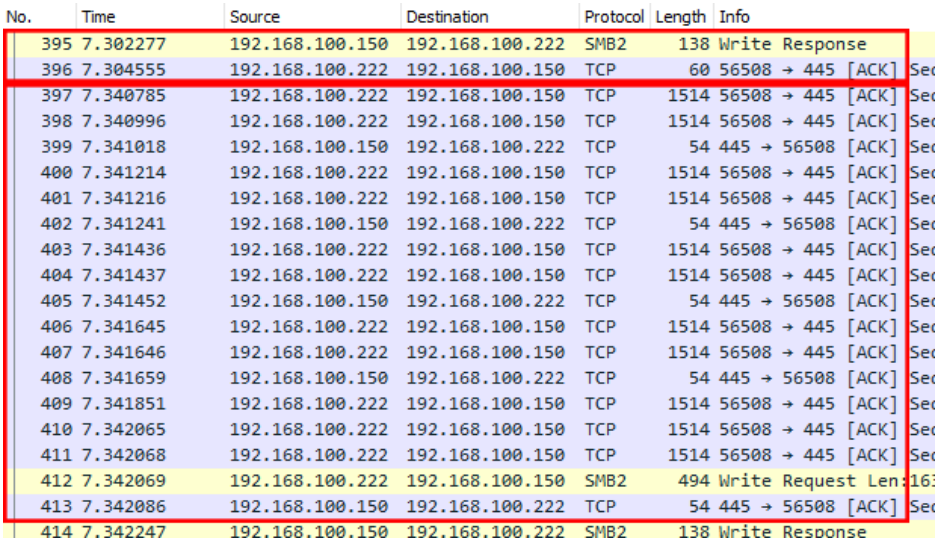
In Abb. 4.3 und 4.4 sind Ausschnitten vom Datenverkehr in Wireshark. Dargestellt ist zuerst

der Write Response, welcher signalisiert der vorherige Request war erfolgreich und danach der nächste Write Request mit neuen Daten entsprechend der Buffergröße.



No.	Time	Source	Destination	Protocol	Length	Info
154	1.738468	192.168.100.150	192.168.100.222	SMB2	138	Write Response
155	1.738759	192.168.100.222	192.168.100.150	TCP	60	56500 → 445 [ACK] Se
159	1.776866	192.168.100.222	192.168.100.150	TCP	1514	56500 → 445 [ACK] Se
160	1.777084	192.168.100.222	192.168.100.150	TCP	1514	56500 → 445 [ACK] Se
161	1.777100	192.168.100.150	192.168.100.222	TCP	54	445 → 56500 [ACK] Se
162	1.777296	192.168.100.222	192.168.100.150	TCP	1514	56500 → 445 [ACK] Se
163	1.777299	192.168.100.222	192.168.100.150	TCP	1514	56500 → 445 [ACK] Se
164	1.777314	192.168.100.150	192.168.100.222	TCP	54	445 → 56500 [ACK] Se
165	1.777509	192.168.100.222	192.168.100.150	TCP	1514	56500 → 445 [ACK] Se
166	1.777511	192.168.100.222	192.168.100.150	SMB2	1062	Write Request Len 81
167	1.777533	192.168.100.150	192.168.100.222	TCP	54	445 → 56500 [ACK] Se

Abbildung 4.3: Wireshark Zeit pro Paket - 8192 Byte Buffer



No.	Time	Source	Destination	Protocol	Length	Info
395	7.302277	192.168.100.150	192.168.100.222	SMB2	138	Write Response
396	7.304555	192.168.100.222	192.168.100.150	TCP	60	56508 → 445 [ACK] Se
397	7.340785	192.168.100.222	192.168.100.150	TCP	1514	56508 → 445 [ACK] Se
398	7.340996	192.168.100.222	192.168.100.150	TCP	1514	56508 → 445 [ACK] Se
399	7.341018	192.168.100.150	192.168.100.222	TCP	54	445 → 56508 [ACK] Se
400	7.341214	192.168.100.222	192.168.100.150	TCP	1514	56508 → 445 [ACK] Se
401	7.341216	192.168.100.222	192.168.100.150	TCP	1514	56508 → 445 [ACK] Se
402	7.341241	192.168.100.150	192.168.100.222	TCP	54	445 → 56508 [ACK] Se
403	7.341436	192.168.100.222	192.168.100.150	TCP	1514	56508 → 445 [ACK] Se
404	7.341437	192.168.100.222	192.168.100.150	TCP	1514	56508 → 445 [ACK] Se
405	7.341452	192.168.100.150	192.168.100.222	TCP	54	445 → 56508 [ACK] Se
406	7.341645	192.168.100.222	192.168.100.150	TCP	1514	56508 → 445 [ACK] Se
407	7.341646	192.168.100.222	192.168.100.150	TCP	1514	56508 → 445 [ACK] Se
408	7.341659	192.168.100.150	192.168.100.222	TCP	54	445 → 56508 [ACK] Se
409	7.341851	192.168.100.222	192.168.100.150	TCP	1514	56508 → 445 [ACK] Se
410	7.342065	192.168.100.222	192.168.100.150	TCP	1514	56508 → 445 [ACK] Se
411	7.342068	192.168.100.222	192.168.100.150	TCP	1514	56508 → 445 [ACK] Se
412	7.342069	192.168.100.222	192.168.100.150	SMB2	494	Write Request Len:16
413	7.342086	192.168.100.150	192.168.100.222	TCP	54	445 → 56508 [ACK] Se
414	7.342247	192.168.100.150	192.168.100.222	SMB2	138	Write Response

Abbildung 4.4: Wireshark Zeit pro Paket - 16384 Byte Buffer

Zwischen Response und Request wird auf der SPS, von der SD Karte, die nächsten Daten gelesen. In Tab. 4.3 wurden die Zugriffszeiten aus den Bildern ermittelt. Dies lässt erkennen, die Zugriffszeit auf die SD-Karte hängt hauptsächlich vom Zugriff selbst ab, nicht aber von der Länge dieser Daten. Während die Übertragung selbst mit der Länge skaliert und nur einen kleinen Teil einnimmt.

Dies erklärt auch die starke Verbesserung der Übertragungszeit mit Vergrößerung des Buffers. Dadurch müssen nämlich seltener Daten von SD-Karte gelesen werden.

Tabelle 4.3: Zeit der Übertragung und SD-Karte

Buffer in Byte	Übertragung in s	Zugriff SD-Karte in s
8192	0.000667	0.038107
16384	0.001301	0.036230

4.2 Vergleich mit anderen Methoden der Dateiübertragung

In diesem Abschnitt wird auf die andere Methode der Übertragung eingegangen.

4.2.1 FTP Client

Der FTP-Client ist derzeit eine viel genutzte Art Dateien zu übertragen. Es gibt viele verschiedene Implementationen. Die hier verglichene ist aus der freien OSCAT Bibliothek. Aber auch Hersteller wie Wago oder Beckhoff bieten welche an.

Vorteile

Stabile Implementierung
Aus der größeren Auswahl und der länger existierenden Lösungen können hiermit stabil Dateien übertragen werden

Speicherverbrauch
Durch ein einfaches Protokoll und damit einfache Authentifizierung beim Server ist der Speicherverbrauch gerade bei kleinen Steuerungen besser.

Nachteile

FTP-Server
Es wird eine FTP-Server Software benötigt, die gerade bei Windows nicht zum Standard gehört, daher muss diese zusätzlich installiert und eingerichtet werden.

Übertragungsgeschwindigkeit
Wie die Tests ergaben ist die Übertragungsgeschwindigkeit gerade bei großen Dateien schlechter.

5 Zusammenfassung und Ausblick

Im letzten Kapitel wird ein zusammenfassender Rückblick auf die Arbeit gegeben. Zusätzlich werden Ansatzpunkte für mögliche Verbesserungen genannt.

5.1 Zusammenfassung

In dieser Arbeit wurde das SMB2-Protokoll auf einer CoDeSys SPS implementiert. Zu Beginn wurden einige Anforderungen aufgestellt, die mindestens erfüllt werden sollten. Es mussten Dateien in beide Richtungen übertragbar sein, und eine Liste der Ordner und Dateien sollte angezeigt werden.

Daraufhin wurden die Grundlagen dazu erarbeitet, ganz zum Anfang stand TCP, ohne das keine Daten übertragen werden können. Anschließend wurde der Aufbau und die Kommunikation des SMB2-Protokolls aus den Microsoft Dokumenten, und deren Abhängigkeiten zu anderen Protokollen, erarbeitet. Zusätzlich musste sich für eine Authentifizierungsmethode entschieden und die Funktion verstanden werden. NTLM wurde ausgewählt aufgrund der leichteren Implementierbarkeit.

Nach Einarbeitung in die verschiedenen Protokolle und aufstellen einer groben Struktur wurde das Projekt erstellt. Die Implementierung in einem eigenen wiederverwendbaren Bausteins erfolgte in Strukturiertem Text. Es war wichtig dass der spätere Anwender nur einen Baustein aufrufen muss, in dem alles nötige passiert. Dadurch, und durch eine einfach Schnittstelle des Bausteins, ist es möglich, dass die Implementierung auch für Anwender, welche nicht tief in der Materie stecken, benutzbar ist.

In der Erprobung stellte sich heraus, dass die Implementierung, auf den getesteten Rechnern, stabil läuft. Die Performance wurde mit verschiedenen Dateien getestet und einem FTP Client gegenübergestellt. Dabei wurde festgestellt, dass die Übertragung mittels SMB2 in allen Fällen schneller war.

Die Implementierung könnte für Anwender interessant sein, welche ihre Dateien nicht an einen großen Firmenserver schicken, sondern eher an Arbeitsplatz PCs, mit welchen auch die Anlage gesteuert wird. Die Frage nach Übertragung, und dem wie, von Dateien stellt sich Erfahrungsgemäß immer öfter. Auch wenn manche Betriebssysteme von Steuerungen

schon etwas ähnliches, das anlegen von Netzlaufwerken, unterstützen ist dies nicht flexibel genug. Außerdem ist der SMB2 Baustein auch auf Steuerungen ohne Windows oder Linux Betriebssystem einsetzbar, solange TCP Funktionen unterstützt werden.

5.2 Technische Verbesserungen

5.2.1 Stabilität und Funktion

Aufgrund vorhandener Hardware und Betriebssystemen wurde die Funktion nur mit zwei verschiedenen Rechner getestet. Im nächsten Schritt müsste die Funktion auf vielen verschiedenen Rechnern getestet werden. Auch konnten bei der Authentifizierung nicht alle möglichen Kombinationen aus beliebig langen Benutzernamen, Passwörtern und Computernamen berücksichtigt werden.

5.2.2 Andere Hersteller

Durch die begrenzten Mittel wurden nur für die Raspberry PI Laufzeitumgebung entwickelt. Allerdings soll die Implementierung auf möglichst vielen Steuerungen lauffähig sein. Hierzu müssen die TCP Funktionen gekapselt werden um sie leicht mit denen der neuen Steuerung austauschen zu können.

5.2.3 Eigene Bibliothek

Es ist sinnvoll aus dem gesamten Projekt eine eigene Bibliothek zu machen. Dies vereinfacht den Einsatz und man hat nicht mehr alle Bausteine im Projekt, welche eigentlich nur intern genutzt werden. Der einzige Baustein, den man dann direkt im Projekt sieht wäre SMB2_Client. Der Benutzer wird dadurch eine einfache Einbindung in verschiedene eigene Projekte ermöglicht.

5.2.4 Authentifizierung

Eine Überlegung ist es die Authentifizierungsmethoden auszuweiten. Dazu ist es aber nötig sich mit den anderen Methoden zu beschäftigen. Vorteil wäre natürlich mehr Sicherheit als bei NTLM. Dies bedeutet allerdings höherer Speicherverbrauch und eventuell auch eine höhere Zykluszeit. Sogar in den Tests zwischen Windows 10 Rechner wurde nur NTLM genutzt, obwohl anderen Methoden verfügbar waren.

Literaturverzeichnis

[1] BORMAN, D. ; BRADEN, B. ; JACOBSON, V. ; SCHEFFENEGGER, R.: *TCP Extensions for High Performance*. 2014. – URL https://www.ietf.org/rfc/rfc7323.txt

[2] C. NEUMAN AND T. YU AND S. HARTMAN AND K. RAEBURN: *The Kerberos Network Authentication Service (V5)*. 07.2017. – URL https://tools.ietf.org/html/rfc4120

[3] KRAWCZYK, Hugo ; BELLARE, Mihir ; CANETTI, Ran: *HMAC: Keyed-Hashing for Message Authentication*. 1997. – URL https://www.ietf.org/rfc/rfc2104.txt

[4] MICROSOFT CORPORATION: *NT LAN Manager (NLMP) Authentication Protocol*. 15.09.2017. – URL https://winprotocoldoc.blob.core.windows.net/productionwindowsarchives/MS-NLMP/[MS-NLMP]-170915.pdf

[5] MICROSOFT CORPORATION: *Server Message Block (SMB) Protocol Versions 2 and 3*. 15.09.2017. – URL https://winprotocoldoc.blob.core.windows.net/productionwindowsarchives/MS-SMB2/[MS-SMB2]-170915.pdf

[6] MICROSOFT CORPORATION: *File System Control Codes*. 2017. – URL https://winprotocoldoc.blob.core.windows.net/productionwindowsarchives/MS-FSCC/[MS-FSCC].pdf

[7] POSTEL, Jon: *Transmission Control Protocol*. 1981. – URL https://www.ietf.org/rfc/rfc793.txt

[8] RIVEST, Ronald L.: *The MD4 Message-Digest Algorithm*. 1992. – URL https://www.ietf.org/rfc/rfc2104.txt

[9] RIVEST, Ronald L.: *The MD5 Message-Digest Algorithm*. 1992. – URL https://www.ietf.org/rfc/rfc1321.txt

Anhang

Der Anhang dieser Arbeit befindet sich auf der beigefügten CD und kann beim betreuenden Professor eingesehen werden.

Der Inhalte im Anhang liegen in folgender Struktur vor:

Bachelorarbeit
: Hier liegt die komplette Arbeit im PDF Format

Abbildungen
: Hier liegen die verwendeten Abbildungen der Bachelorarbeit, wenn in der Arbeit nicht anders angegeben handelt es sich dabei um Eigenanfertigungen.

Quellen
: Hier liegen die Quellen, falls Online verfügbar, als PDF oder Textdatei.

Projektdateien
: Hier liegt das archivierte CoDeSys Projekt.

Programmdokumentation
: Hier befinden sich die CoDeSys Bausteine im PDF Format.

Testedateien
: Diese Dateien wurden zum testen der Übertragung verwendet.

Beispiel Kommunikation
: Hier liegen Wireshark Mitschnitte der Dateiübertragung und des Explorers. Diese können mit Wireshark (https://www.wireshark.org/) betrachtet werden.

Glossar

FTP File Transfer Protocol, ein Dateiübertragungsprotkoll für Ethernetnetzwerke

Little Endian Das niedrigste Byte wird an der kleinsten Speicheradresse gespeichert

Network Byte Order Big-Endian Format, das höchswertige Byte wird an der kleinsten Speicheradresse gespeichert

NTLM NT Lan Manager, Authentifizierungsmethode welche bei SMB2 genutzt wird.

SMB2 Server Message Block 2, Protokoll für Datei- und Druckdienste.

SPS Speicherprogrammierbare Steuerung, meist ein eigenständiges Gerät welches Steuerungsaufgaben übernimmt

Versicherung über die Selbstständigkeit

Hiermit versichere ich, dass ich die vorliegende Arbeit im Sinne der Prüfungsordnung nach §16(5) APSO-TI-BM ohne fremde Hilfe selbstständig verfasst und nur die angegebenen Hilfsmittel benutzt habe. Wörtlich oder dem Sinn nach aus anderen Werken entnommene Stellen habe ich unter Angabe der Quellen kenntlich gemacht.

Hamburg, 12. Februar 2018	
Ort, Datum	Unterschrift