Abschlussprüfung Sommer 2025

Fachinformatiker für Anwendungsentwicklung

Dokumentation zur betrieblichen Arbeit

Die Entwicklung eines Wireshark-Dissectors für das BAOS-Kommunikationsprotokoll

Abgabetermin: 21. März 2025

Prüfungsbewerber

Adam Rigely
[ADRESSE]

Projektbetreuer

[NAME]
[ADRESSE]

Ausbildungsbetrieb

[NAME]
[ADRESSE]

Inhaltsverzeichnis

Abkürzungsverzeichnis

B

BACnet · *Building Automation and Control Networks*
BAOS · *Bus Access and Object Server*

H

HTTPS · *Hypertext Transfer Protocol Secure*

I

IDE · *Integrated Development Environment*

K

KNX · *Konnex*

R

RX · *Receiver*

S

SSH · *Secure Shell*

T

Tvb · *Testy, Virtual(-izable) Buffer*
TX · *Transmitter*

U

USB · *Universal Serial Bus*

V

VS-Code · *Visual Studio Code*

1. Einleitung

In der vorliegenden Arbeit wird die Abwicklung des betrieblichen Abschlussprojekts dokumentiert, das im Rahmen einer Umschulung zum Fachinformatiker für Anwendungsentwicklung vom Autor durgeführt wird. Mit dem innerbetrieblichen Projekt wurde der Autor von seinem Ausbilder beauftragt.

1.1. Projektumfeld

Die betriebliche Projektarbeit wird im Rahmen eines Praktikums bei dem Unternehmen [FIRMENNAME] durchgeführt. Das Unternehmen ist ein Anbieter von Automationssystemen in der Gebäudetechnik. Den Kunden werden alle Aspekte des Automatisierungsprozesses von Gebäuden angeboten, wie bspw. die Ausarbeitung von Anlagenkonzepten, sowie die Installation und Wartung von Gebäudeautomationssystemen.

1.2. Projekthintergrund

1.2.1. Integration des BAOS- / KNX-Kommunikationsprotokolls

Zu der Automatisierung von Gebäuden werden Zonenregler angeboten, die verschiedene Kommunikationsprotokolle, wie z.B. BACnet und Modbus, unterstützen. [FIRMENNAME] möchte zukünftig das Kommunikationsprotokoll KNX in den Zonenreglern implementieren. Um dieses Ziel zu erreichen, werden das Kommunikationsprotokoll BAOS und ein Weinzierl KNX BAOS Modul verwendet.

Das BAOS-Protokoll ist eine quelloffene Abstrahierung des KNX-Protokolls, das, mit einem geeigneten KNX BAOS Modul, eine einfache und schnelle Integration des eher komplizierten KNX-Protokolls ermöglicht. Sowohl das BAOS-Kommunikationsprotokoll, als auch das Weinzierl KNX BAOS Modul wurden von der Weinzierl Engineering GmbH entwickelt.

1.2.2. Funktionsbeschreibung des BAOS-Protokolls / -Moduls

An der Entwicklung der hauseigenen Implementierung des BAOS-Protokolls wird derzeit bei [FIRMENNAME] im Rahmen eines größeren Projekts gearbeitet. Nachdem dieses Kommunikationsprotokoll in den Zonenreglern integriert wird, wird durch das BAOS-Modul die Kommunikation mit KNX-Geräten ermöglicht. Das BAOS-Modul wird in diesem System als ein Übersetzer fungieren, der die BAOS-Telegramme vom Zonenregler in KNX-Telegramme konvertiert und an die KNX-Geräte übergibt, sowie die KNX-Telegramme in BAOS-Telegramme konvertiert und an den Zonenregler übergibt.

1.2.3. Aktueller Stand

Das BAOS-Protokoll und ein BAOS-Modul werden zwar zukünftig in den eigenen Zonenreglern implementiert, allerdings sind noch einige Anpassungen erforderlich, damit dieses Kommunikationsprotokoll in den Geräten verwendet werden kann. Die Entwicklung des BAOS-Protokolls erfolgt auf einem PC mit Windows 11, der mit einem Weinzierl KNX BAOS Modul über eine USB-Schnittstelle verbunden ist. Im Rahmen der Entwicklung werden testweise einfache KNX-fähige Geräte, wie z.B. Taster, von dem PC über das BAOS-Modul kontrolliert.

1.3. Projektbegründung

Während der Entwicklung der angepassten Version des BAOS-Protokolls ist das regelmäßige Testen wesentlich, damit Fehler und unerwünschte Situationen vermieden werden können. Tests werden mit Hilfe des Netzwerkanalyse-Tools Wireshark durchgeführt. Da das BAOS-Protokoll in Wireshark zurzeit nicht unterstützt wird (Stand in der Version 4.4.5), wird die BAOS-Payload des USB-Paketes nicht zerlegt, sondern als eine Menge reiner, unstrukturierter hexadezimaler Werte dargestellt. Der aktuelle Stand wird in *Anlage 1* präsentiert.

Es ist zwar möglich, während des Testens des BAOS-Kommunikationsprotokolls und der Analyse der Pakete die Daten als einzelne Bytes manuell zu interpretieren, dieses Vorgehen nimmt jedoch unnötig viel Zeit in Anspruch. Um dies zu optimieren fiel die Entscheidung auf die Entwicklung eines Wireshark-Dissectors. Die übersichtliche Darstellung der Pakete und deren Inhalt kann das Testen beschleunigen und die Erkennung eventueller Fehler verbessern.

1.4. Projektziel

Ziel des Projekts ist, für Wireshark ein Dissector-Plugin zu entwickeln, das Wireshark in die Lage versetzt, BAOS-Telegramme erkennen, analysieren und strukturiert darstellen zu können. Da die Kommunikation über das BAOS-Protokoll sowohl in dem aktuellen Entwicklungsstand, als auch in der zukünftigen Produktionsumgebung über eine serielle Verbindung erfolgt, werden die BAOS-Telegramme in jedem Fall in einem FT 1.2-Frame eingebettet. Daher ist die Zerlegung des Frames ebenfalls Bestandteil des BAOS-Dissectors.

1.5. Projektschnittstellen

Der Entwickler-PC, worauf Wireshark läuft, empfängt die Daten von dem BAOS-Modul über eine USB-Schnittstelle. Wireshark wird es mit Hilfe des Plugins *USBPcap* ermöglicht, diese Kommunikation mitzuschneiden. Nach dem Erhalt des Mitschnitts wird ein geeigneter, für das jeweilige Protokoll registrierter Dissector von Wireshark aufgerufen. Wireshark stellt eine große Auswahl von Lua-API-Funktionen für die Entwicklung eines Dissectors zur Verfügung. Bei der Registrierung des BAOS-Dissectors und des BAOS-Protokolls und der Entwicklung des Dissectors werden diese verwendet.

An dem Projekt haben sich der Auftraggeber und der Autor beteiligt. Mit dem Auftraggeber wurden die Anforderungen an den BAOS-Dissector besprochen. Das Projekt wurde jedoch ausschließlich vom Autor realisiert.

1.6. Projektabgrenzung

Obwohl bei [FIRMENNAME] an mehreren BAOS-bezogenen Projekten gearbeitet wird, wird dieses Projekt auf die Entwicklung des BAOS-Dissectors für Wireshark beschränkt.

2. Projektplanung

2.1. Konkretisierung der Anforderungen

Als erstes müssen die konkreten Anforderungen an das Projekt gesetzt werden. Im Rahmen einer Besprechung mit dem Auftraggeber wurden die Erwartungen an das BAOS-Dissector-Plugin in der Form eines Lastenhefts beschrieben. In diesem Dokument wurden die groben Voraussetzungen zusammengestellt, die das fertige Produkt erfüllen muss. Dies wird als *Anlage 2* an die Projektdokumentation angehängt.

2.2. Entwurf eines Projektplans

Nachdem das „*was*“ konkretisiert wurde, muss das „*wie*“ geplant werden. Um das Projekt effektiv durchführen zu können, wurde es strukturiert in Phasen geteilt. Da das iterative Testen während der Programmierung in diesem Projekt von Vorteil ist, fiel die Entscheidung auf das erweiterte Wasserfallmodell. Der Grund dafür besteht in den verschiedenen BAOS-Subdiensten, dies wird in dem Kapitel „Implementierungsphase“ näher erläutert.

Um den Aufbau des Projekts bzw. der Projektphasen, sowie die Zeitplanung mit den geschätzten erforderlichen Stunden übersichtlich veranschaulichen zu können, wurde ein Projektablaufplan entworfen. Der wird als *Anlage 3* an die Projektdokumentation angehängt.

2.3. Ressourcen

2.3.1. Hardware-Ressourcen

Wie es in der Einleitung bereits erwähnt wurde, ist dieses Projekt mit dem größeren BAOS-Projekt verwandt. Da es sich um die Optimierung des Testverfahrens handelt, wurden grundsätzlich die gleichen Hardware-Ressourcen verwendet, wie bei der Entwicklung des Kommunikationsprotokolls. Das bedeutet, dass die Hardware-Ressourcen bereits vor dem Beginn des vorliegenden Projekts zur Verfügung standen. Eine Auflistung der Hardware- sowie der Software-Ressourcen können der *Anlage 4* entnommen werden.

2.3.2. Software-Ressourcen

Die Entwicklung des Dissector-Plugins für das Netzwerkanalyse-Tool Wireshark wurde unter dem Betriebssystem Microsoft Windows 11 Pro, in dem Code-Editor Microsoft Visual Studio Code durchgeführt. Für die Versionsverwaltung wurde Git bzw. GitHub verwendet. Pläne, Diagramme und Mockups wurden in Canva entworfen. Für das Schreiben einer Produktdokumentation wurde das Markdown-Tool Obsidian genutzt.

2.4. Projektkosten

Angesichts, dass die Hardware-Ressourcen bereits vorhanden waren, wurden deren Beschaffungskosten in der Kostenberechnung des Projekts nicht berücksichtigt.

Der erste Faktor bei der Berechnung der Kosten sind die Humanressourcen. Das Projekt wurde ausschließlich von dem Praktikanten durchgeführt. Der Auftraggeber beauftragte den Autor mit dem innerbetrieblichen Projekt und wurde zudem zur Klärung der Anforderungen konsultiert. Außerdem nahm er das Softwareprodukt nach Abschluss des Projekts ab.

Aus datenschutzrechtlichen Gründen wurde bei den beiden Personen mit dem durchschnittlichen Stundenlohn eines ausgebildeten Anwendungsentwicklers gerechnet.

Die meisten während der Projektdurchführung verwendeten Softwareprodukte waren entweder bereits vorhanden oder waren kostenlos nutzbar. Eine kostenpflichtige Version von Canva wurde jedoch genutzt.

Für alle weiteren Ressourcen (bspw. Strom, Abnutzung der Hardware) wurde ein Pauschalbetrag von brutto 5€ pro Tag genommen, um die Berechnung einfach zu halten. An einem Arbeitstag wurde durchschnittlich 4 Stunden an dem Projekt gearbeitet.

Aus diesen Angaben und der für das Projekt gesetzten 80 Stunden ergeben sich die untenstehende Berechnung und groben Projektkosten von brutto 1928,30€.

	Zeitdauer	Kosten	Kosten gesamt
Stundensatz Praktikant	80 Stunden	22,15 € / Stunde	1.772,00 €
Stundensatz Ausbilder	2 Stunden	22,15 € / Stunde	44,30 €
Canva Pro Abonnement	1 Monat	12,00 € / Monat	12,00 €
Pauschalbetrag Weitere Ressourcen (Strom, Hardware Abnutzung etc...)	80 Std / 4 Std = 20 Tage	5 € / Tag	100,00 €
1.928,30 €			

Berechnung der Projektkosten

2.5. Recherche nach der Entwicklung eines Dissectors

Obwohl generelle Grundkenntnisse über Wireshark-Dissectoren vorhanden waren, bestand keine praktische Erfahrung bei der Entwicklung eines solchen Plugins. Aus diesem Grund wurde eine Recherche zu diesem Thema durchgeführt.

Bei der Entwicklung boten sich die Programmiersprache C und die Scriptsprache Lua an. Es wurde bereits vor dem Projektbeginn beschlossen, dass der Dissector in Lua entwickelt wird. Die wesentlichsten Gründe dafür sind:

1. die Syntax von Lua ist signifikant einfacher als die von C, was einen positiven Einfluss auf die Entwicklungsdauer hat,
2. die Lua-Schnittstellen zu Wireshark sind einfach gestaltet worden, was die Entwicklung vereinfacht und die Dauer ebenso verkürzt,
3. die Integration eines Lua-Plugins in Wireshark kann schneller erledigt werden, als die eines C-Plugins,
4. Lua-Scripte werden nicht kompiliert, was schnelle Testprozesse während der Entwicklung ermöglicht.

Aufgrund dieser Entscheidung zielte die Recherche auf den generischen Aufbau eines Wireshark-Dissectors und die von Wireshark zur Verfügung gestellten Lua-Schnittstellen.

2.6. Pflichtenheft

Nach der Recherche wurden genügend Kenntnisse gesammelt, um auf das Lastenheft eine fachliche Antwort in der Form eines Pflichtenhefts geben zu können. Anhand dieser Kenntnisse und der BAOS-Protokollbeschreibung wurden für die in dem Lastenheft bestehenden groben Anforderungen konkrete Lösungen geplant. Das Pflichtenheft wird als *Anlage 5* an die Projektdokumentation angehängt.

2.7. Einrichtung der Entwicklungsumgebung

Als letzter Vorbereitungsschritt in der Planungsphase wurde eine für die Entwicklung geeignete Umgebung eingerichtet.

Da eine vollständige IDE nicht benötigt wurde, fiel die Entscheidung auf den Code-Editor Microsoft Visual Studio Code, der auf dem Entwickler-PC installiert wurde. VS-Code ist bekannt für seinen geringen Ressourcenbedarf und wurde für genau diese Eigenschaft für das Projekt gewählt. Ein Lua-Language-Server-Plugin für den Code-Editor wurde ebenso installiert, was die Entwicklung vereinfacht und beschleunigt.

Schließlich wurde ein Lua-Interpreter auf dem Entwickler-PC eingerichtet, der ermöglichte, Lua-Funktionen während der Entwicklung unabhängig von Wireshark testen zu können.

3. Entwurf

3.1. Mockup

Als der erste Schritt in der Entwurfsphase wurde ein Mockup entworfen, in dem präsentiert wird, wie die BAOS-Pakete, nach der Implementierung des Dissector-Plugins, in Wireshark angezeigt werden sollen. Die Darstellung wird von Wireshark intern erledigt, indem eine Baum-Struktur für die Pakete und deren Inhalt in dem Dissector-Plugin konstruiert wird. Für das Zeichnen des Mockups wurde das Design-Tool Canva genutzt. Dieses Mockup wird als *Anlage 6* an die Projektdokumentation angehängt.

3.2. Aufbau eines Wireshark-Dissectors

Während der Entwicklung eines Wireshark-Dissectors werden im Wesentlichen von Wireshark vordefinierte Bausteine verwendet. Diese werden genutzt, um eine hierarchische Baum-Struktur in Wireshark anzeigen zu lassen.

3.2.1. Proto

Ein grundlegendes Element in jedem Wireshark-Dissector ist ein Proto-(Protokoll)-Objekt. Mit Hilfe eines solchen Elements kann ein neues Protokoll in Wireshark registriert werden. Ein Proto-Objekt ermöglicht außerdem die Filterung der Pakete nach Protokollen in Wireshark und es fungiert als Basis bei der Strukturierung eines Protokoll-Baums.

3.2.2. TreeItem

TreeItems (Baum-Objekte) sind ebenso wesentliche Elemente bei der Strukturierung eines Protokoll-Baums. Die können als übergeordnete Elemente in der Baum-Struktur betrachtet

werden, die weitere, zu dem jeweiligen TreeItem gehörige Elemente beinhalten – die eventuell wiederum als TreeItems funktionieren können.

3.2.3. ProtoField

ProtoFields (Protokollfelder) sind Elemente in der Protokoll-Baum-Struktur, welche die in den Paketen bestehenden eigentlichen Daten repräsentieren. ProtoFields werden TreeItems zugewiesen. Falls eine komplexere Baum-Struktur benötigt wird, können ProtoFields gegebenenfalls wiederum als weitere TreeItems fungieren.

3.2.4. Tvb

Tvb-Objekte repräsentieren die Datenpakete, die an Wireshark übergeben werden. Bereiche und gewisse Bytes in den Tvb-Objekten können ProtoFields und TreeItems zugewiesen werden, damit diese die korrekten Daten in der Baum-Struktur darstellen.

3.2.5. ProtoExpert

ProtoExperts sind Info-Felder die TreeItems zugewiesen werden können. Die können den Benutzern hilfreiche Informationen zur Verfügung stellen, welche während der Paketanalyse nützlich werden können. Ein weiterer Vorteil bei der Nutzung eines ProtoExperts ist, dass Pakete, die ProtoExpert-Objekte beinhalten, werden in Wireshark gefärbt hervorgehoben, wodurch die Filterung solcher Telegramme noch einfacher wird. Die Farbe wird von Wireshark dem Parameter *„Severity“* entsprechend automatisch gewählt. Dieser wird bei der Definition des ProtoExpert-Objekts angegeben. Ein ProtoExpert mit dem Schwierigkeitsgrad *„WARN“* wird bspw. gelb, ein ProtoExpert mit *„ERROR“* rot angezeigt. Die in Wireshark definierten Schwierigkeitsgraden sind in der Wireshark-Dokumentation erläutert.

3.3. Planung des BAOS-Dissectors

3.3.1. Zerlegung

Anhand der genannten Bausteine und der BAOS-Protokollbeschreibung wurde ein Plan für den BAOS-Dissector entworfen. Bei dem Entwerfen des Aufbaus und der Logik des Dissectors wurde nicht nur die Zerlegung der BAOS-Payload, sondern auch des FT 1.2-Frames geplant. Ebenfalls musste für die weitere Zerlegung des Telegramms der jeweilige BAOS-Subdienst beachtet werden.

3.3.2. Integrität

Bei der Erstellung des Plans war die Integrität der Pakete ein weiterer wichtiger Faktor. Es wurde bereits vor dem Projekt festgestellt, dass einige an Wireshark übergebene Pakete nicht vollständig sind. Es ist kein Teil des Projekts um für dieses Problem eine Lösung zu finden, es ist allerdings wichtig, um solche empfangenen Pakete korrekt zerlegen zu können. Deshalb wurde bei der Planung des Dissectors dieser Fakt in Betracht gezogen.

3.3.3. Heuristischer Dissector

Bei den meisten Netzwerk-Kommunikationsprotokollen besteht die Möglichkeit um bei der Registrierung des Dissectors für ein gewisses Protokoll eine Netzwerk-Portnummer zu

verwenden, wie bspw. Port 22 bei SSH, Port 443 bei HTTPS, oder Port 47808 bei BACnet. Da das BAOS-Protokoll derzeit über eine USB-Verbindung verwendet wird, kann zur Erkennung des Protokolls eine Portnummer nicht genutzt werden. Damit BAOS-Pakete erkannt werden können, wurde ein sogenannter heuristischer Dissector entwickelt. Im Gegensatz zu einem konventionellen Wireshark-Dissector wird anstatt einer Portnummer ein (oder mehrere) Muster verwendet um ein Protokoll identifizieren zu können. Die an Wireshark übergebenen USB-Pakete werden analysiert und falls das vordefinierte Muster gefunden wird, schreitet der Dissector mit der Zerlegung fort. Im Falle des BAOS-Dissectors werden der FT 1.2-Frame-Header und das BAOS-Mainservice-Byte als Erkennungsmuster verwendet. Da dieses Muster in jedem seriellen BAOS-Paket vorhanden ist, und es extrem unwahrscheinlich ist, dass genau dieses Muster in einem nicht-BAOS-Paket vorkommt, ist es die beste Alternative in diesem Fall.

3.3.4. Aktivitätsdiagramm

Um eine optimale Übersicht über den Aufbau sowie die Funktionsweise des Dissectors zu gewähren, wurde ein UML-Aktivitätsdiagramm erstellt. Aufgrund des Umfangs des Plans wurde das Diagramm in zwei Teile zerlegt. In dem ersten Diagramm wird die Logik der Überprüfung der Pakete dargestellt, die vor der eigentlichen Zerlegung nötig ist. In dem zweiten Diagramm wird die Zerlegung der Pakete präsentiert. Die Diagramme werden als *Anlage 7* bzw. *Anlage 8* an die Projektdokumentation angehängt.

4. Implementierung

4.1. Anschluss des Lua-Dissectors an Wireshark

Wie es im Kapitel „Planungsphase“ bereits erwähnt wurde, ist ein wesentlicher Vorteil von Lua, dass Lua-Plugins einfach in Wireshark implementiert werden können. Der BAOS-Wireshark-Dissector, wie die meisten Lua-Wireshark-Dissectoren, besteht aus einer einzigen *.lua*-Datei, die in das *plugins*-Verzeichnis von Wireshark kopiert werden muss. Wireshark prüft das Verzeichnis beim Start und lädt die verfügbaren Plugins.

4.2. Definition des Protokolls

In einem Wireshark-Dissector wird als erstes das Protokoll definiert. Das erfolgt durch die Erzeugung eines Proto-Objektes mit Hilfe der *Proto.new*-Funktion. An die Funktion werden ein langer und ein kurzer Protokollname als Parameter übergeben. Der lange wird bei der Darstellung des Protokolls und der Pakete verwendet, während der kurze Filterfunktionen erfüllt.

4.3. Definition der *dissector*-Funktion

Nachdem das BAOS-Protokoll definiert wurde, musste die *dissector*-Funktion des Proto-Objekts definiert werden. Diese Funktion wird bei dem Aufruf eines Wireshark-Dissectors aufgerufen.

Im Falle des BAOS-Dissectors wird in der *dissector*-Funktion nicht nur die eigentliche Zerlegung des BAOS-Paketes definiert, sondern ebenfalls die Zerlegung des FT 1.2-Frames.

Außerdem muss der Dissector zuerst eine grobe Überprüfung der Daten durchführen um zu erkennen, ob es sich um einen BAOS-Frame handelt, da die Erkennung heuristisch erfolgt.

4.4. Suche nach Muster

Bei der Überprüfung ob es sich um eine BAOS-Payload in einem FT 1.2-Frame handelt, wird an den ersten 5 Stellen nach dem Startbyte des FT 1.2-Frames gesucht, da vor dem eigentlichen Frame gelegentlich noch andere Daten stehen. Nachdem das Startbyte gefunden wurde, werden noch weitere Positionen überprüft. Als detaillierte Illustration des Aufbaus und der Erkennung der BAOS-Telegramme, wird *Anlage 9* an die Projektdokumentation angehängt.

4.5. Überprüfung der Integrität

In der *dissector*-Funktion wird als nächstes die Integrität des Paketes überprüft. Wie bereits erwähnt, sind einige an Wireshark übergebene BAOS-Pakete nicht vollständig. Wenn der BAOS-Dissector solche Pakete erkennt, sollen diese trotzdem zerlegt und zusätzlich mit einem ProtoExpert-Objekt markiert werden.

Es wird überprüft, ob die Länge des Tvb-Objektes, in dem sich die an Wireshark übergebene Daten befinden, mindestens so lang ist, wie die anhand der Länge-Byte kalkulierte Länge des FT 1.2-Frames. Wenn dieser Schritt erfolgt, wird es überprüft, ob am Ende des Frames das FT 1.2-Ende-Byte gefunden wird. Falls es festgestellt wird, dass der Frame unvollständig ist, wird dem FT 1.2-Baum ein ProtoExpert-Objekt hinzugefügt, um die Benutzer über die Unvollständigkeit zu benachrichtigen.

4.6. Erstellung des Paket-Baums

Nach den oben beschriebenen Prüfprozessen können die ersten Schritte der eigentlichen Paket-Zerlegung durchgeführt werden.

Um die Werte der einzelnen Bytes, deren Bedeutungen, und die dazugehörigen Beschriftungen darzustellen, wird die Struktur des Paket-Baums erstellt. Der *dissector*-Funktion wird, unter anderem, ein TreeItem als Parameter übergeben, das als Basis der Baumstruktur verwendet wird. Die ProtoFields, die diesem Objekt hinzugefügt werden, werden als Teile der Baumstruktur bei BAOS-Paketen angezeigt.

4.7. FT 1.2-Header

Um die in dem *Mockup von der Darstellung der BAOS-Pakete* illustrierte gewünschte Baumstruktur zu erstellen, wird dem Basis-TreeItem das BAOS-Protokoll Proto-Objekt hinzugefügt. Dieses Feld repräsentiert das ganze Paket, vom Startbyte bis zum Endbyte. Damit der Baum-Aufbau dem wahren Aufbau eines seriellen BAOS-Paketes entspricht, wird unter dem Protokoll-TreeItem ein weiterer TreeItem namens *„FT 1.2 Frame“* angelegt. Innerhalb des FT-1.2-Frame-Baums wird für den FT 1.2-Header ein TreeItem erstellt, der die entsprechenden ersten 5 Bytes des ganzen Paketes repräsentiert.

Nachdem die Basis-Struktur erstellt wurde, wird der Frame-Header zerlegt und dessen Inhalt, als einzelnen Bytes, angezeigt. Für eine ordentliche Darstellung werden ProtoFields verwendet. Diese ermöglichen eine nähere Definition, wie genau die Bytes in dem Paket

dargestellt werden sollen und was sie bedeuten. In dem ProtoField wird für das jeweilige Feld auch ein kurzer Name vergeben, der Filterung anhand gewisser Werte in dem vorliegenden Feld ermöglicht. Um den beschriebenen Aufbau zu veranschaulichen, werden Screenshots von dem Quellcode an die Projektdokumentation angehängt. In *Anlage 10* und *Anlage 11* werden der Aufbau der Baumstruktur des Headers, bzw. die Definition der ProtoFields präsentiert.

4.8. Zerlegung der BAOS-Payload

Nach dem FT 1.2-Header wird die BAOS-Payload in dem Frame zerlegt. Für diesen Teil wird wieder ein neuer TreeItem innerhalb des FT 1.2-Frame-Baums angelegt. Während der BAOS-Mainservice-Code sich in jedem BAOS-Paket, mit dem gleichen Wert befindet, hängt die Struktur des restlichen Teils der BAOS-Payload von dem vorliegenden Subservice ab. Deshalb muss erst der Subservice-Code gelesen, und die Zerlegung dementsprechend weitergeführt werden.

In dem BAOS-Protokoll sind insgesamt 10 Subdienste definiert und 8 davon haben sowohl eine Sender- (Request) als auch eine Empfangsvariante (Response), daher müssen insgesamt 18 Telegrammvarianten behandelt werden. Für die Zerlegung der verschiedenen Telegrammarten werden Funktionen programmiert, die je nach dem Subservice-Code aufgerufen werden. In den Funktionen werden die Bytes in der BAOS-Payload der BAOS-Protokoll-Dokumentation entsprechend ausgewertet und anhand definierter ProtoFields dem BAOS-Baum hinzugefügt. Ein Screenshot von der Auflistung der Zerlegungsfunktionen in der Form einer Lua-Tabelle wird an die Projektdokumentation als *Anlage 12* angehängt.

Exemplarisch wird darüber hinaus ein Screenshot von einer Zerlegungsfunktion an die Projektdokumentation als *Anlage 13* angehängt, um die grundlegende Logik der Zerlegung zu illustrieren.

4.9. Zerlegung unvollständiger Frames

Während der Entwicklung der Funktionen hat sich die Entscheidung für das erweiterte Wasserfallmodell vorteilhaft ausgewirkt, da jede Zerlegungsfunktion nach deren Programmierung einzeln getestet werden konnte. Bei den iterativen Tests traten *„Range is out of bounds“* Fehlermeldungen mehrmals auf. Der Grund dafür ist, dass gelegentlich unvollständige Frames in Wireshark empfangen werden. Bei solchen Telegrammen kann es vorkommen, dass der Dissector auf Bytes an Stellen zugreifen will, die in dem Tvb-Objekt nicht existieren. Um Fehler zu vermeiden, müssen unvollständige BAOS-Telegramme korrekt behandelt werden.

Die implementierte Lösung für dieses Problem ist der Vergleich der Länge des Tvb-Objekts mit dem Index des Bytes worauf der Dissector zugreifen will. Somit kann sichergestellt werden, dass die gesuchte Stelle in dem Array vorhanden ist. Diese Überprüfung wird bei jedem Zugriff auf ein Byte in der Payload-Zerlegungsphase und der FT 1.2-Trailer-Zerlegungsphase vorgenommen. Bei der Zerlegung des Headers ist die Überprüfung nicht nötig, da sie erst startet sobald genügend Bytes empfangen wurden.

Ein Beispiel einer solchen Überprüfung bei dem Hinzufügen von ProtoFields wird in der *Anlage 14* illustriert.

4.10. FT 1.2-Trailer

Nach der Zerlegung der BAOS-Payload muss die Zerlegung des FT 1.2-Trailers programmiert werden. Für den Trailer wird ein TreeItem in dem FT 1.2-Baum erstellt. Dem Baum werden das FT 1.2-Endbyte und die Checksumme des Frames hinzugefügt. Während das Endbyte in jedem Paket den gleichen Wert hat, ändert sich die Checksumme allerdings den in dem Frame bestehenden Daten entsprechend. Obwohl es ursprünglich nicht geplant gewesen war, da es kaum weiteren Aufwand verursachte, wurde nach Absprache mit dem Auftraggeber für die Implementierung einer Checksumme-Überprüfung-Funktion entschieden.

Eine Checksumme-Berechnung wird durchgeführt und bei einem Checksumme-Fehler dem Trailer ein ProtoExpert hinzugefügt um die Benutzer über das Problem zu benachrichtigen. Ein Beispiel wie ein Checksumme-Fehler in einem BAOS-Telegramm in Wireshark dargestellt wird, wird als *Anlage 15* an die Projektdokumentation angehängt.

5. Testen

Während der Entwicklung des BAOS-Dissectors wurden die Zerlegungsfunktionen zwar separat getestet, allerdings musste der fertige Wireshark-Dissector auch als komplettes Softwareprodukt getestet werden. Die Grundlagen für die Testphase waren das *Lastenheft* und das *Pflichtenheft*. Um feststellen zu können, dass der Dissector den Erwartungen entspricht, wurden die in den erwähnten Dokumenten gestellten Voraussetzungen als Benchmark genutzt.

Während der Testphase wurden White-Box-Tests durchgeführt. Sowohl die vom PC an das BAOS-Modul gesendeten Befehle als auch die vom BAOS-Modul an den PC gesendeten Antworten wurden von strukturellen sowie inhaltlichen Aspekten überprüft. Im BAOS-Protokoll wurden für sämtliche Subdienste Testfunktionen entwickelt. Mit Hilfe dieser Funktionen waren verschiedene Telegramme von allen Subdiensten an das BAOS-Modul gesendet worden, auf die es antwortete. Um den Aufbau einer solchen Funktion zu illustrieren, wird ein Screenshot von dem Quellcode an die Projektdokumentation als *Anlage 16* angehängt.

Die Kommunikation anhand der erwähnten Telegramme wurde überwacht. Unter anderem wurde kontrolliert, ob die BAOS-Pakete erkannt werden, ob nicht-BAOS-Pakete nicht fehlerhaft als BAOS-Pakete interpretiert werden, ob die BAOS-Subdienste richtig zerlegt werden, ob der Paket-Baum korrekt aufgebaut wird, ob die ProtoFields den korrekten Bytes zugewiesen werden und ob die ProtoFields die korrekten Inhalte haben. Des Weiteren wurde darauf geachtet, ob unvollständige Frames und Pakete mit einer fehlerhaften Checksumme erwartungsgemäß behandelt werden.

Die Testphase wurde erfolgreich abgeschlossen und der Dissector entsprach den Erwartungen. Ein Screenshot wie ein BAOS-Paket nach der Implementierung des BAOS-Dissectors dargestellt wird, wird als *Anlage 17* an die Projektdokumentation angehängt.

Nach der Testphase wurde der BAOS-Dissector dem Auftraggeber abgegeben. Er war mit den Ergebnissen zufrieden und hat das Wireshark-Plugin abgenommen. Das Projekt durfte somit in die Dokumentationsphase eintreten.

6. Dokumentation

Für das Schreiben der Produktdokumentation wurde das Markdown-Tool Obsidian verwendet. In der wird detailliert erklärt, wie das Wireshark-Dissector-Plugin installiert, konfiguriert und genutzt werden kann. Anhand der Dokumentation werden zukünftige Einsätze des Dissectors auf weiteren Geräten erleichtert. Auf Barrierefreiheit wurde geachtet, indem neben der deutschen Version ebenso ein englisches Dokument geschrieben wurde. Die deutsche Produktdokumentation wird an die Projektdokumentation als *Anlage 18* angehängt.

Schließlich wurde für das Projekt die vorliegende Projektdokumentation geschrieben.

7. Fazit

7.1. Soll- / Ist-Vergleich

Die am Anfang des Projekts geplanten Stunden konnten im Wesentlichen eingehalten werden. Minimale Abweichungen kamen in der Planungsphase vor. Der Grund dafür bestand in der Komplexität des Themas, weshalb die geplante Recherche eine Stunde länger dauerte. Die Erstellung des Lasten- und des Pflichtenhefts nahm allerdings weniger Zeit in Anspruch, da die Anforderungen vom Anfang an relativ klar waren und die Dokumente hauptsächlich einer schriftlichen Klarstellung dienten. Da die Differenzen einander kompensieren konnten, blieb die Planungsphase bei 8, bzw. die Projektdurchführungsdauer bei 80 Stunden.

7.2. Gewonnene Kenntnisse

Die wichtigsten Kenntnisse die durch die Durchführung des Projekts gewonnen wurden, können in drei Hauptpunkten zusammengefasst werden:

1. Die gesammelten Erfahrungen bei der Entwicklung des Dissectors werden zukünftig bei der Erstellung weiterer Dissectoren nützlich sein,
2. Die gesammelten Erfahrungen mit Lua werden die Entwicklung anderer Dissectoren ebenso vereinfachen und beschleunigen,
3. Die gesammelten Erfahrungen während der Projektabwicklung werden die Durchführung zukünftiger Projekte erleichtern.

7.3. Ausblick

Der Wireshark-Dissector für das BAOS-Protokoll kann derzeit während der Entwicklung der angepassten Version des Protokolls verwendet werden, Anpassungen werden allerdings zukünftig nötig. Zurzeit wird der Entwickler-PC mit dem BAOS-Modul über die Entwicklungsplatine verbunden, die eine einfache Verbindungsmöglichkeit über eine USB-Schnittstelle ermöglicht. Nachdem das BAOS-Modul und das BAOS-Protokoll in den Zonenreglern implementiert werden, wird das Mitlauschen auf dem Bussystem aber neue Lösungen benötigen.

Literaturverzeichnis

Ierusalimschy, R. (2016). *Programming in Lua.* Rio de Janeiro: Roberto Ierusalimschy.

IHK Koblenz. (01. 10 2022). *Umsetzung Abschlussprüfung Teil2 in den IT-Berufe.* Von IHK Koblenz Offizielle Website: https://www.ihk.de/blueprint/servlet/resource/blob/5689110/0b10d73808a953075587965fe433ac0a/umsetzung-ap-teil-2-neue-it-berufe-2020-data.pdf abgerufen

Lamping, U., Sharpe, R., & Warnicke, E. (24. 02 2025). *Wireshark Developer's Guide 4.5.0*. Von Wireshark Offizielle Website: https://www.wireshark.org/download/docs/Wireshark%20User%27s%20Guide.pdf abgerufen

Stepstone. (2025). *Fachinformatiker/in Anwendungsentwicklung Gehälter in Deutschland*. Von Stepstone: https://www.stepstone.de/gehalt/Fachinformatiker-in-Anwendungsentwicklung.html abgerufen

Touch, J., Lear, E., Ono, K., Eddy, W., Trammell, B., Iyengar, J., . . . Nishida, Y. (25. 02 2025). *Service Name and Transport Protocol Port Number Registry*. Von IANA - Internet Assigned Numbers Authority: https://www.iana.org/assignments/service-names-port-numbers/service-names-port-numbers.xhtml abgerufen

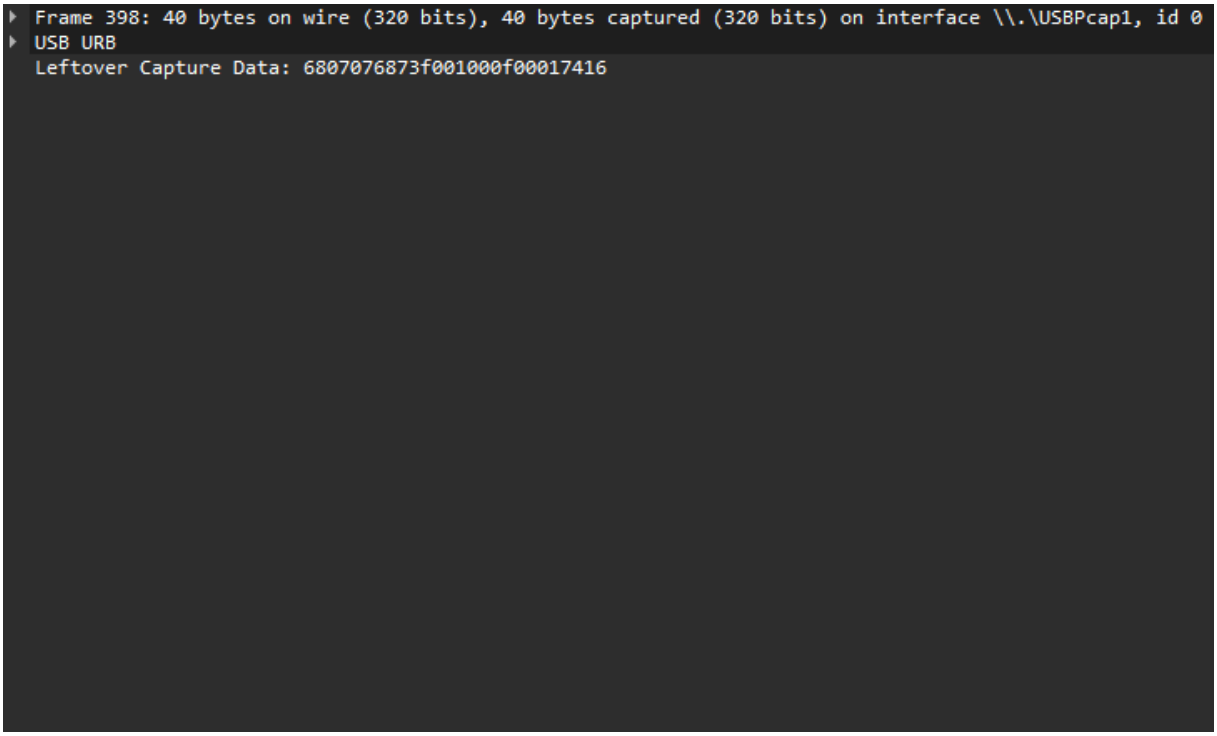
Weinzierl Engineering GmbH. (26. 07 2017). *The World of BAOS*. Von Weinzierl Engineering GmbH Offizielle Website: https://weinzierl.de/images/download/development/82x/the_world_of_baos.pdf abgerufen

Weinzierl Engineering GmbH. (22. 07 2024). *KNX BAOS Binary Protocol.* Von Weinzierl Engineering GmbH Offizielle Website: https://weinzierl.de/images/download/documents/baos/weinzierl-knx-baos-binaryprotocol-v2.pdf abgerufen

Wireshark Foundation. (kein Datum). *About Wireshark*. Von Wireshark Offizielle Website: https://www.wireshark.org/about.html abgerufen

Anlagen

Anlage 1. Aktueller Stand in Wireshark ohne das Dissector-Plugin



Aktueller Stand in Wireshark ohne das Dissector-Plugin

Anlage 2. Lastenheft

1. Funktionale Anforderungen an das Wireshark-Plugin

Mit Hilfe des Plugins muss Wireshark in die Lage versetzt werden, um:

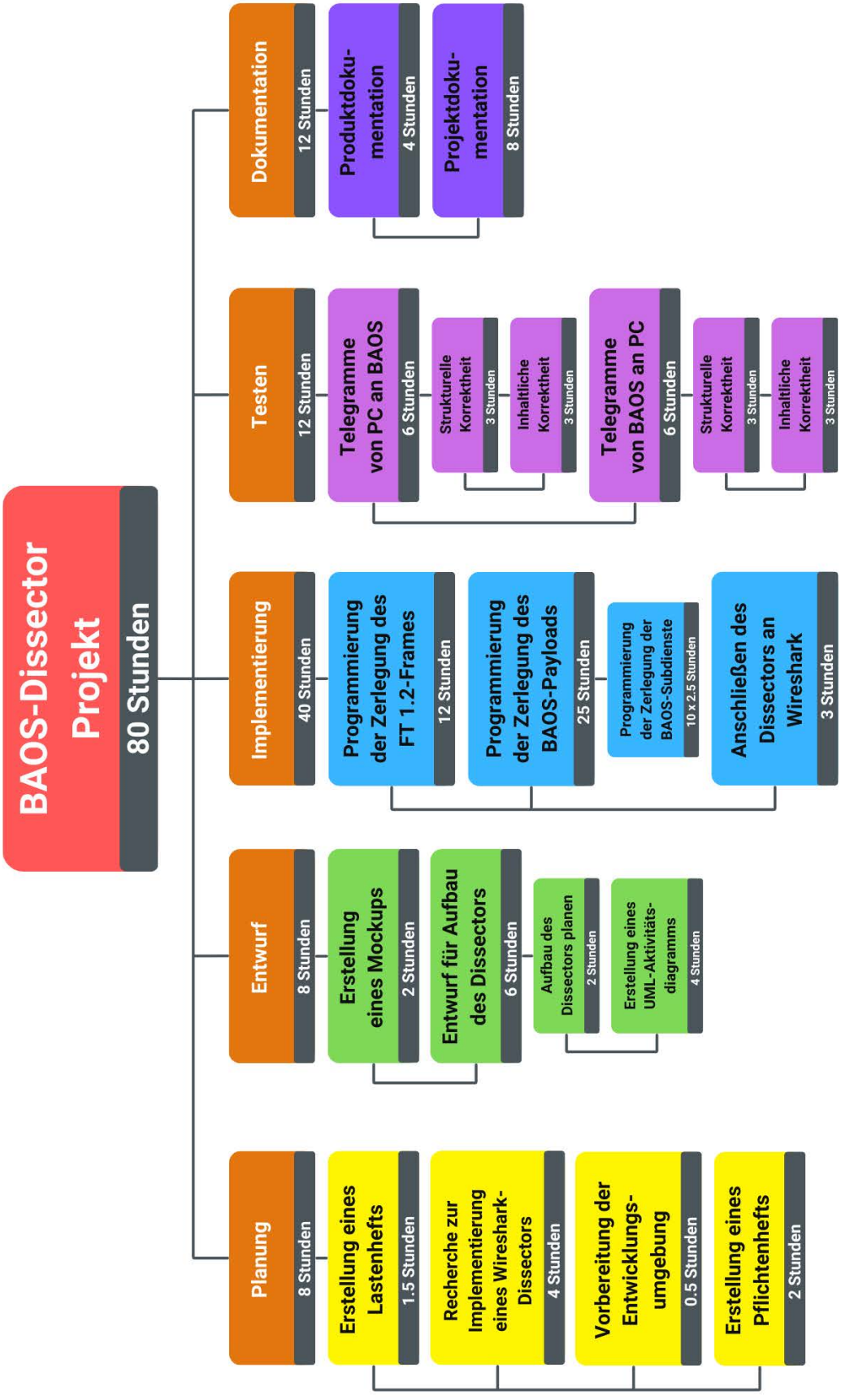
- 1.1. BAOS-Pakete (TX sowie RX) erkennen zu können,
- 1.2. zwischen den BAOS-Subdiensten unterscheiden zu können,
- 1.3. die einzelnen Teile der BAOS-Subdienste erkennen zu können,
- 1.4. die einzelnen Teile der BAOS-Pakete strukturiert darstellen zu können,
- 1.5. den FT 1.2-Frame, in dem die BAOS-Payload eingebettet ist, erkennen zu können,
  - 1.5.1. den FT 1.2-Header erkennen zu können,
  - 1.5.2. den FT 1.2-Trailer erkennen zu können,
- 1.6. die einzelnen Teile des FT 1.2-Frames strukturiert darstellen zu können,
- 1.7. die Filterung der BAOS-Telegramme zu ermöglichen,
- 1.8. die Filterung anhand gewisser Werte im Paket zu ermöglichen.

2. Nicht funktionale Anforderungen an das Wireshark-Plugin

Ferner ist es erwartet von dem Plugin, dass:

- 2.1. es leicht in Wireshark zu integrieren ist,
- 2.2. es portabel ist.

Anlage 3. Projektablaufplan



Projektablaufplan

Anlage 4. Ressourcen

1. Hardware

1.1. Entwickler-PC

- Intel Core i5-10400
- Intel UHD Graphics 630
- Kingston SA400S37240G 240
- 8 GB Arbeitsspeicher

1.2. Weinzierl KNX BAOS Modul 832

1.3. Weinzierl BAOS Development Board

1.4. MDT SCN-IP000.02 – KNX IP Interface

1.5. MDT STV-0640.02 – KNX Bus Netzteil

1.6. MDT BE-GTT4S.01 – KNX Glastaster

1.7. MDT-SCN-RT1GS.01 – KNX Glas Raumtemperaturregler

2. Software

2.1. Microsoft Windows 11 Pro

2.2. Microsoft Visual Studio Code

2.3. Lua-Language-Server-Plugin für VS-Code von *sumneko*

2.4. Microsoft Office Professional Plus 2019

2.5. Wireshark

2.6. Git

2.7. Canva

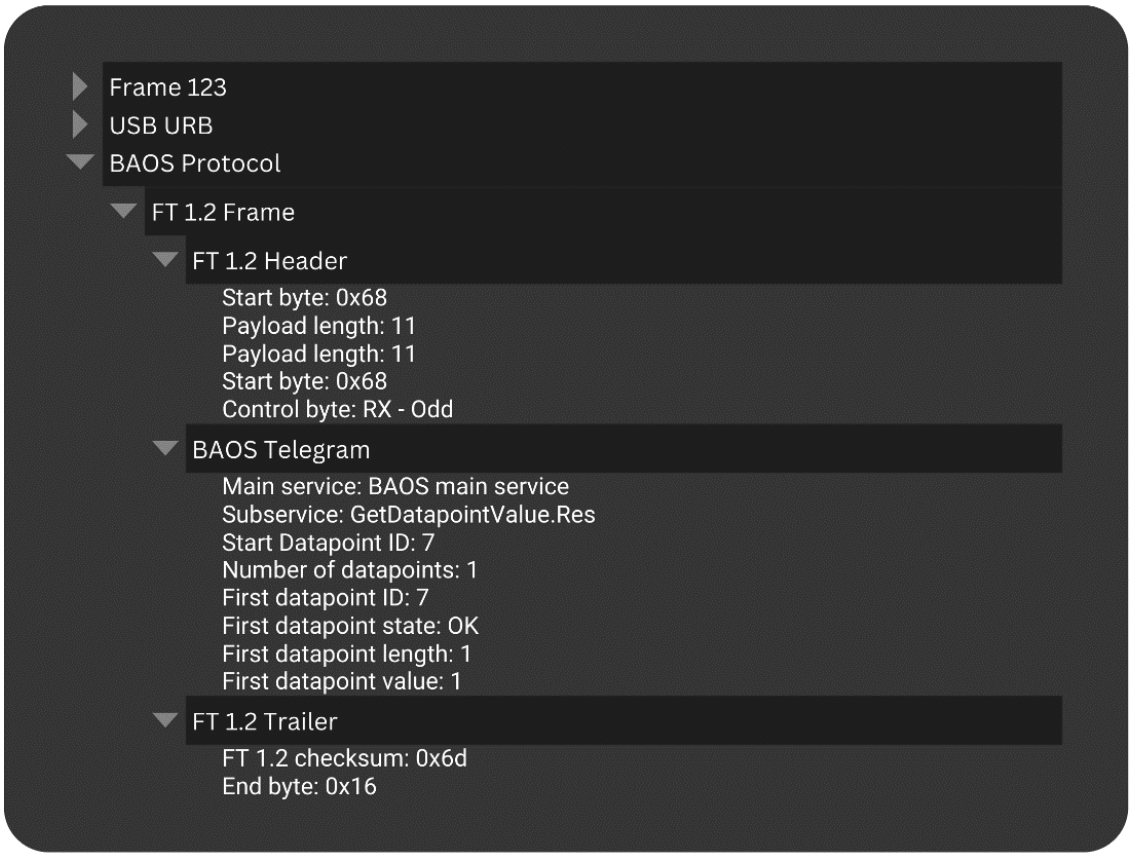
2.8. Obsidian

Anlage 5. Pflichtenheft

Lösung der Anforderungen an das Wireshark-Plugin

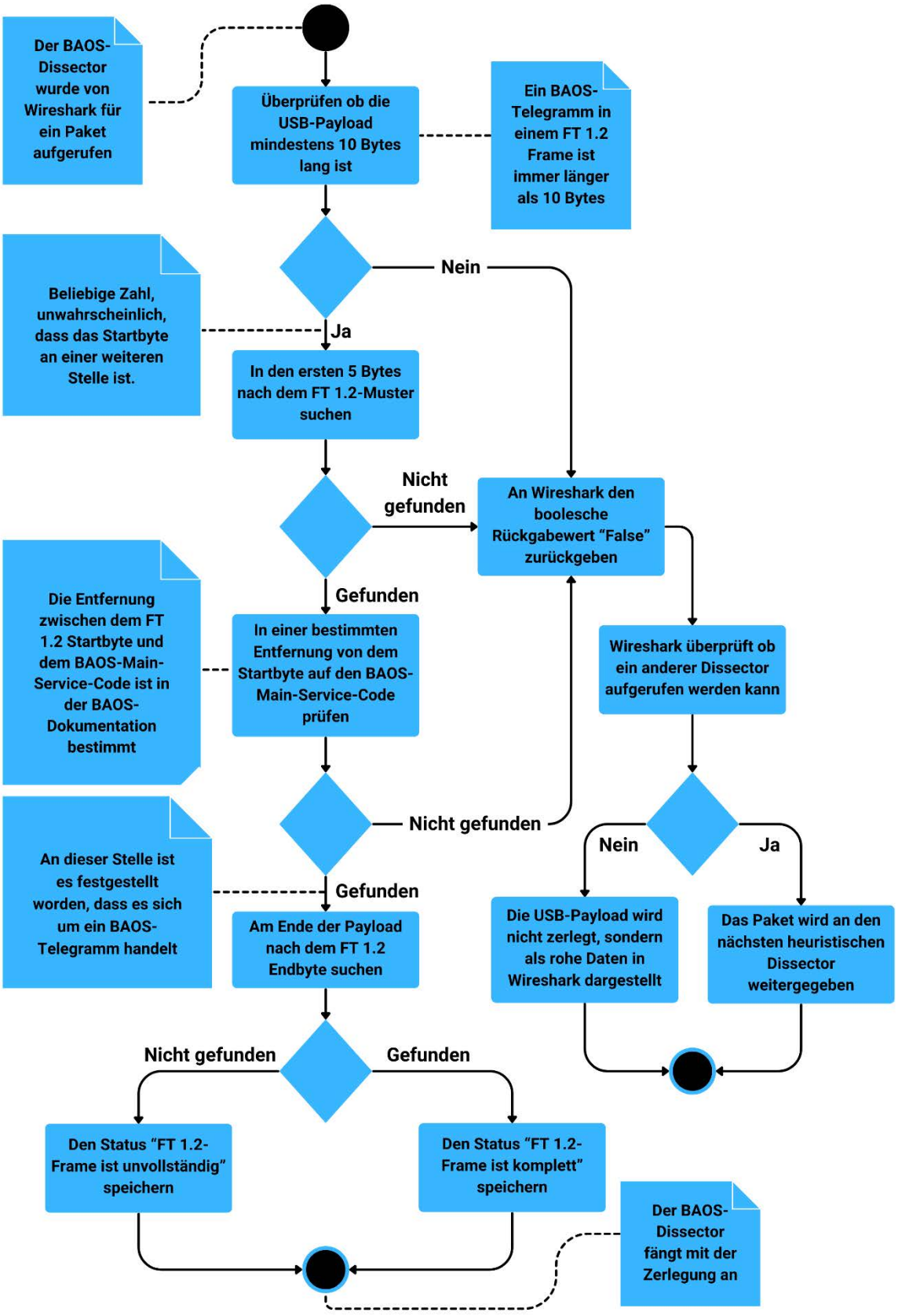
1. Um die an das Wireshark-Plugin gestellten Anforderungen zu lösen, wird ein Dissector-Plugin für Wireshark entwickelt.
2. Das Dissector-Plugin wird in der Skriptsprache Lua entwickelt.
3. Das Plugin wird als ein heuristischer Dissector an Wireshark angeschlossen.
4. Die FT 1.2-Frames werden heuristisch, anhand des einzigartigen Paket-Header-Aufbaus erkannt.
5. Die BAOS-Telegramme werden anhand des einzigartigen BAOS-Mainservice-Codes sowie des bestimmten Offsets in dem Paket erkannt.
6. Der FT 1.2-Header wird anhand des FT 1.2-Startbytes sowie des einzigartigen FT 1.2-Header-Musters erkannt.
7. Die einzelnen Teile des FT 1.2-Headers werden anhand der bestimmten Offsets erkannt.
8. Der FT 1.2-Trailer wird anhand des FT 1.2-Endbytes sowie der bestimmten Offsets erkannt.
9. Zwischen den verschiedenen BAOS-Subdiensten wird anhand der einzigartigen BAOS-Subservice-Codes unterschieden.
10. Für die übersichtliche, strukturierte Darstellung werden die von Wireshark bereitgestellten Lua-API-Funktionen und -Strukturen verwendet.
11. Der FT 1.2-Frame sowie das eingebettete BAOS-Telegramm werden als eine Baum-Struktur aufgebaut und in Wireshark dargestellt.
12. Die einzelnen Teile des FT 1.2-Frames und des BAOS-Telegramms werden als Wireshark-Protokollfelder in der Baum-Struktur dargestellt.
13. Die Filterung der BAOS-Pakete wird anhand des in Wireshark registrierten BAOS-Protokolls ermöglicht.
14. Die Filterung der Pakete mit gewissen Werten wird anhand der registrierten Protokollfelder ermöglicht.
15. Aufgrund der Tatsache, dass der Dissector als ein Lua-Plugin entwickelt wird, werden die einfache Integrierung und Portabilität ermöglicht.

Anlage 6. Mockup von der Darstellung der BAOS-Pakete



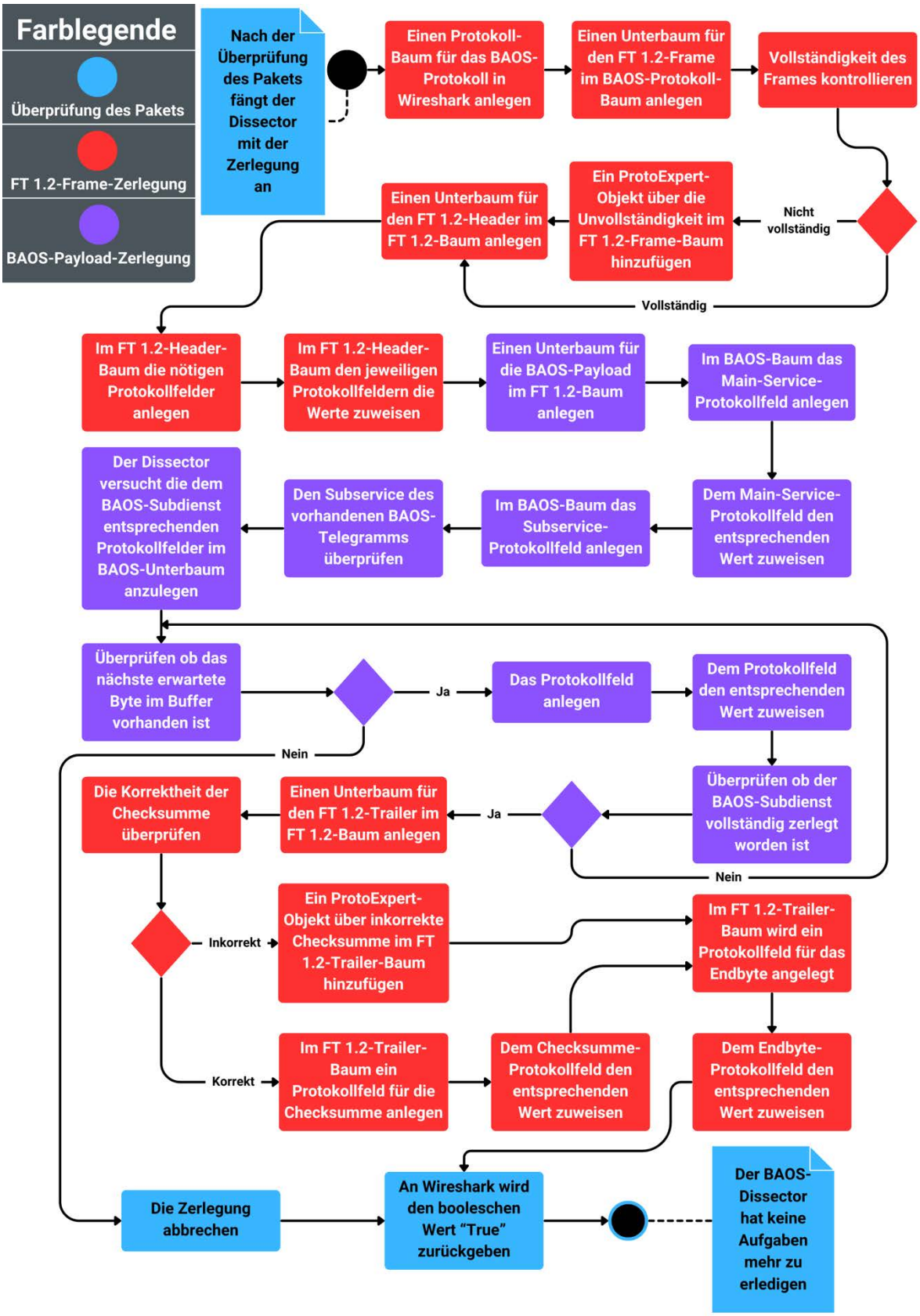
Mockup von der Darstellung der BAOS-Pakete

Anlage 7. UML-Aktivitätsdiagramm der Paketüberprüfung



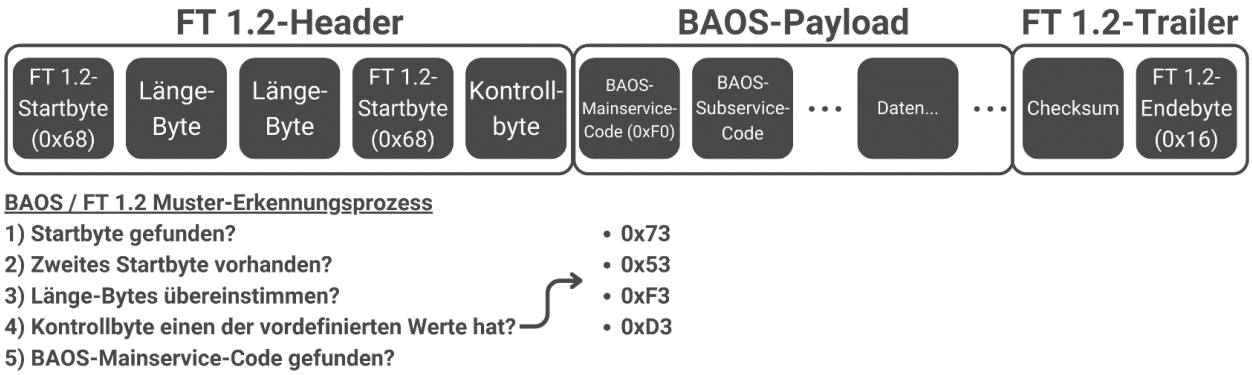
UML-Aktivitätsdiagramm der Paketüberprüfung

Anlage 8. UML-Aktivitätsdiagramm der Paketzerlegung



UML-Aktivitätsdiagramm der Paketzerlegung

Anlage 9. Aufbau und Erkennung eines seriellen BAOS-Telegramms



Aufbau und Erkennung eines seriellen BAOS-Telegramms

Anlage 10. Aufbau des FT 1.2-Header-Baums

```
-- Add FT 1.2 header subtree under the FT 1.2 subtree
local ft12HeaderTree = ft12Tree:add
                                    (
                                        f_ft12Header,
                                        packetBuffer(ft12StartIndex, 5)
                                    )

-- Add FT 1.2 header ProtoFields under the FT 1.2 header subtree
ft12HeaderTree:add
                    (
                        f_ft12StartByte,
                        packetBuffer(ft12StartIndex, 1)
                    )
ft12HeaderTree:add
                    (
                        f_ft12LengthByte,
                        packetBuffer(ft12StartIndex + 1, 1)
                    )
ft12HeaderTree:add
                    (
                        f_ft12LengthByte,
                        packetBuffer(ft12StartIndex + 2, 1)
                    )
ft12HeaderTree:add
                    (
                        f_ft12StartByte,
                        packetBuffer(ft12StartIndex + 3, 1)
                    )
ft12HeaderTree:add
                    (
                        f_ft12ControlByte,
                        packetBuffer(ft12StartIndex + 4, 1)
                    )
```

Aufbau des FT 1.2-Header-Baums

Anlage 11. Definition der FT 1.2-Header-ProtoFields

```
-- FT 1.2 ProtoFields

local f_ft12MainField <const> =
                                    ProtoField.protocol
                                    (
                                        "baos.ft12Frame",
                                        "FT 1.2 frame"
                                    )

local f_ft12Header <const> =
                                    ProtoField.protocol
                                    (
                                        "baos.ft12FrameHeader",
                                        "FT 1.2 frame header"
                                    )

local f_ft12StartByte <const> =
                                    ProtoField.uint8
                                    (
                                        "baos.ft12StartByte",
                                        "Start byte",
                                        base.HEX
                                    )

local f_ft12LengthByte <const> =
                                    ProtoField.uint8
                                    (
                                        "baos.ft12LengthByte",
                                        "Payload length",
                                        base.DEC
                                    )

local f_ft12ControlByte <const> =
                                    ProtoField.uint8
                                    (
                                        "baos.ft12ControlByte",
                                        "Control byte",
                                        base.HEX,
                                        vs_Ft12ControlByte
                                    )
```

Definition der FT 1.2-Header-ProtoFields

Anlage 12. Lua-Table mit Zerlegungsfunktionen

```
-- Table of BAOS subservice dissector functions
-- indexed by the corresponding subservice codes
local subserviceDissectFuncs =
                                {
                                    [0x01] = dissectGetServerItemReq,
                                    [0x02] = dissectSetServerItemReq,
                                    [0x03] = dissectGetDatapointDescriptionReq,
                                    [0x04] = dissectGetDescriptionStringReq,
                                    [0x05] = dissectGetDatapointValueReq,
                                    [0x06] = dissectSetDatapointValueReq,
                                    [0x07] = dissectGetParameterByteReq,
                                    [0x08] = dissectSetParameterByteReq,
                                    [0x81] = dissectGetServerItemRes,
                                    [0x82] = dissectSetServerItemRes,
                                    [0x83] = dissectGetDatapointDescriptionRes,
                                    [0x84] = dissectGetDescriptionStringRes,
                                    [0x85] = dissectGetDatapointValueRes,
                                    [0x86] = dissectSetDatapointValueRes,
                                    [0x87] = dissectGetParameterByteRes,
                                    [0x88] = dissectSetParameterByteRes,
                                    [0xC1] = dissectDatapointValueInd,
                                    [0xC2] = dissectServerItemInd
                                }
```

Lua-Table mit Zerlegungsfunktionen

Anlage 13. Beispiel einer Zerlegungsfunktion

```
local function dissectSetServerItemReq(packetBuffer, packetBufferLen, dataFirstIndex, baosTree)

    -- Variables for readability
    local bufferBytesArray <const> = packetBuffer:bytes()
    local nrOfServerItems = (packetBufferLen >= (dataFirstIndex + 4))
                            and bufferBytesArray:int(dataFirstIndex + 2, 2) or nil

    -- Index offsets used while looping through all server items
    local serverItemIdOffset        = dataFirstIndex + 4
    local serverItemLengthOffset    = serverItemIdOffset + 2
    local serverItemDataOffset      = serverItemLengthOffset + 1

    local serverItemLength = bufferBytesArray:int(serverItemLengthOffset, 1)

    -- Add ID of the starting server item
    if packetBufferLen >= (dataFirstIndex + 2) then
        baosTree:add
                    (
                        f_startServerItemId,
                        packetBuffer(dataFirstIndex, 2)
                    )
    else return false end
    -- Add number of server items
    if packetBufferLen >= (dataFirstIndex + 4) then
        baosTree:add
                    (
                        f_nrOfServerItems,
                        packetBuffer(dataFirstIndex + 2, 2)
                    )
    else return false end
    -- Loop through all server items
    for i = 1, nrOfServerItems, 1 do
        -- Add server item ID
        if packetBufferLen >= (serverItemIdOffset + 2) then
            baosTree:add
                        (
                            f_serverItemId,
                            packetBuffer(serverItemIdOffset, 2)
                        )
        else return false end
        -- Add server item data length
        if packetBufferLen >= (serverItemLengthOffset + 1) then
            baosTree:add
                        (
                            f_serverItemLength,
                            packetBuffer(serverItemLengthOffset, 1)
                        )
        else return false end
        -- Add server item data
        if packetBufferLen >= (serverItemDataOffset + serverItemLength) then
            baosTree:add
                        (
                            f_serverItemData,
                            packetBuffer(serverItemDataOffset, serverItemLength)
                        )
        else return false end

        -- Set offset to start byte of the next server item
        if packetBufferLen >= (serverItemIdOffset + 3 + serverItemLength) then
            serverItemIdOffset = serverItemIdOffset + 3 + serverItemLength
        else break end
    end
end
```

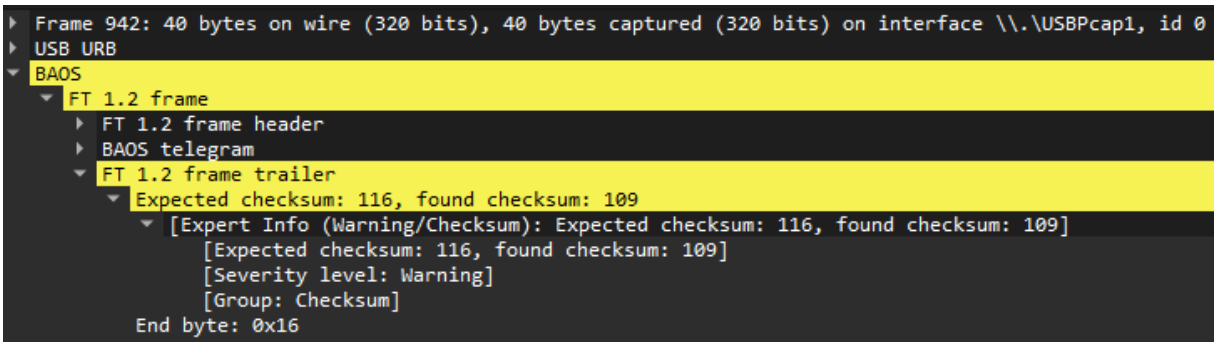
Beispiel einer Zerlegungsfunktion

Anlage 14. Beispiel der Überprüfung des gültigen Tvb-Bereichs

```
-- Add starting datapoint ID
if packetBufferLen >= (dataFirstIndex + 2) then
    baosTree:add
                (
                    f_startDpId,
                    packetBuffer(dataFirstIndex, 2)
                )
else
    return false
end

-- Add number of datapoints
if packetBufferLen >= (dataFirstIndex + 4) then
    baosTree:add
                (
                    f_nrOfDps,
                    packetBuffer(dataFirstIndex + 2, 2)
                )
else
    return false
end
```

Beispiel der Überprüfung des gültigen Tvb-Bereichs

Anlage 15. Beispiel eines Checksumme-Fehlers



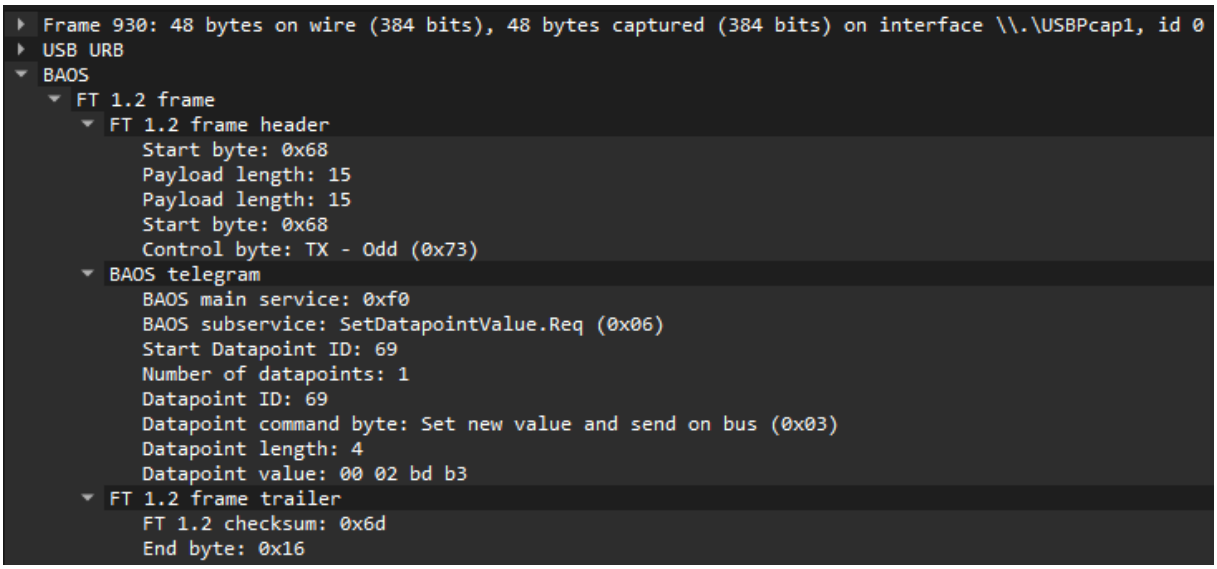
Beispiel eines Checksumme-Fehlers

Anlage 16. Beispiel einer Testfunktion

```
void BaosTester::testSetDatapointValue()
{
    printf("TESTING SET DATAPOINT VALUE STARTED\n\n");

    SetDatapointValue sdv1(1, serialConnection);
    sdv1.setBoolean(true, CommandByte::SetNewValueAndSendOnBus, true);
    sdv1.setBoolean(false, CommandByte::SetNewValueAndSendOnBus, true);

    SetDatapointValue sdv2(2, serialConnection);
    sdv2.setBoolean(true, CommandByte::SetNewValueAndSendOnBus, true);
    sdv2.setBoolean(false, CommandByte::SetNewValueAndSendOnBus, true);

    SetDatapointValue sdv3(3, serialConnection);
    sdv3.setBoolean(true, CommandByte::SetNewValueAndSendOnBus, true);
    sdv3.setBoolean(false, CommandByte::SetNewValueAndSendOnBus, true);

    SetDatapointValue sdv4(4, serialConnection);
    sdv4.setBoolean(true, CommandByte::SetNewValueAndSendOnBus, true);
    sdv4.setBoolean(false, CommandByte::SetNewValueAndSendOnBus, true);

    SetDatapointValue sdv5(9, serialConnection);
    sdv5.setFloatValue2Byte(3.14, CommandByte::SetNewValueAndSendOnBus, true);

    SetDatapointValue sdv6(11, serialConnection);
    sdv6.setSignedValue2Byte(25, CommandByte::SetNewValueAndSendOnBus, true);

    SetDatapointValue sdv7(13, serialConnection);
    sdv7.setUnsignedValue2Byte(13975, CommandByte::SetNewValueAndSendOnBus, true);

    SetDatapointValue sdv8(17, serialConnection);
    sdv8.setFloatValue4Byte(17.97528, CommandByte::SetNewValueAndSendOnBus, true);

    SetDatapointValue sdv9(25, serialConnection);
    sdv9.setSignedValue4Byte(-12759, CommandByte::SetNewValueAndSendOnBus, true);

    SetDatapointValue sdv10(69, serialConnection);
    sdv10.setUnsignedValue4Byte(179635, CommandByte::SetNewValueAndSendOnBus, true);

    printf("\nTESTING SET DATAPOINT VALUE FINISHED\n\n");
}
```

Beispiel einer Testfunktion

Anlage 17. Darstellung eines BAOS-Pakets mit dem BAOS-Dissector



Darstellung eines BAOS-Pakets mit dem BAOS-Dissector

Anlage 18. Produktdokumentation

Übersicht

Dies ist eine kurze Anleitung, die bei der Einrichtung einer Wireshark-Umgebung mit dem BAOS-Dissector-Plugin helfen soll. Nachdem die Umgebung eingerichtet wird, wird Wireshark in der Lage sein, USB-Pakete mit Payloads, die das BAOS-Protokoll nutzen, zerlegen zu können.

Diese Anleitung nimmt an, dass Wireshark bereits installiert worden ist, deshalb ist eine Installationsanleitung zu Wireshark kein Teil dieser Anleitung. Falls Hilfe bei der Installation von Wireshark nötig sein sollte, besuche bitte ihre offizielle Webseite.

Was ist Wireshark?

> *"Wireshark ist das führende Tool für Netzwerk-Protokoll-Analyse auf der Welt. Es ermöglicht, es sehen zu können, was im eigenen Netzwerk auf mikroskopischer Ebene geschieht. Es ist der de facto (und oftmals de jure) Standard in vielen Branchen und Bildungseinrichtungen."*
>
> Übersetztes Zitat von der Wireshark-Website (20.03.2025)

Was ist BAOS?

> *"BAOS – kurz für "Bus Access and Object Server" – ist eine universelle Architektur um KNX-Konnektivität für eine große Auswahl von Produkten zu ermöglichen. In der Produktpalette von KNX-BAOS-Lösungen werden von Weinzierl eine Menge skalierbarer Module und leistungsfähiger Geräte angeboten, die eine komplette Integration von Anwendungen in das KNX-System rapid ermöglichen."*
>
> Übersetztes Zitat aus dem Dokument *"The world of BAOS"* (20.03.2025)

Was ist ein Dissector?

Dissectoren sind ein wichtiger Teil von Wireshark. Sie ermöglichen die Zerlegung von Frames und Paketen, die bestimmten Protokolle verwenden. Ein Dissector kann entweder als ein integraler Teil von Wireshark, oder als ein Plugin entwickelt werden. Im letzteren Fall ist die modifizierung des Quellcodes von Wireshark nicht nötig. Die Entwicklung eines Dissectors wird von Wireshark entweder in C oder in Lua unterstützt.

Dieses BAOS-Dissector-Plugin wurde in Lua entwickelt.

Wie wird der Dissector installiert

Dies ist eine Schritt-für-Schritt-Anleitung für die Installation des BAOS-Dissector-Plugins.

> **Anmerkung**
>
> Der Dissector wurde mit der Version 4.4.5 von Wireshark entwickelt und getestet . Der Dissector wird zwar mit einer anderen Version wahrscheinlich ebenso funktionieren, es ist jedoch **nicht garantiert**. Während der Nutzung einer anderen Version ist es möglich, dass die Beschreibung der Schritte für deine Arbeitsumgebung nicht vollständig passt.

Vorbereitungen

Vergewissere dich, dass:

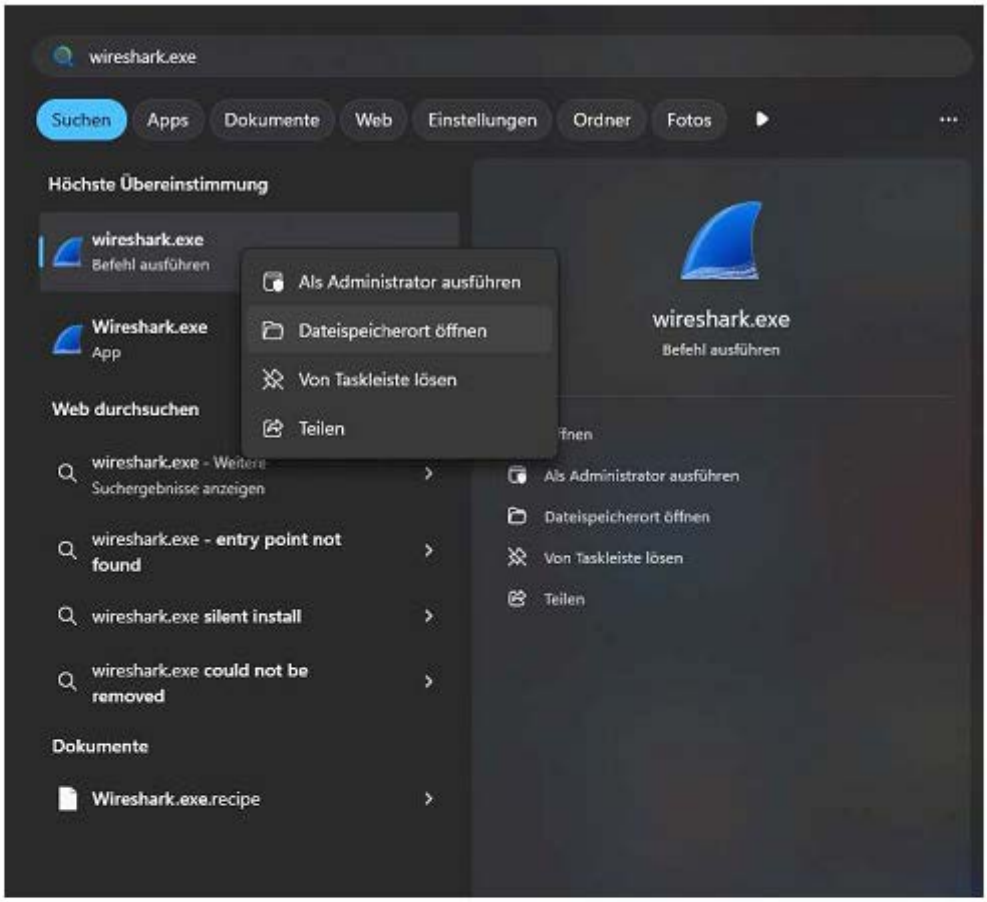
- Du Wireshark erfolgreich installiert hast
- Du über die BAOS-Dissector-*.lua*-Datei verfügst

Installation des Dissectors

Das Installationsverzeichnis finden

Du musst erst das Installationsverzeichnis von Wireshark finden

1. Drücke die *Windows*-Taste
2. Gib `wireshark.exe` in das Suchfeld ein
3. Klicke mit der rechten Maustaste auf die gefundene *wireshark.exe*-Datei
4. Selektiere *Dateispeicherort öffnen*



An dieser Stelle sollst du dich in dem Hauptverzeichnis von Wireshark in einem neuen Windows Explorer-Fenster finden.

Den Dissector in sein Zielverzeichnis kopieren

Nun musst du die BAOS-Dissector-*.lua*-Datei in ihr Zielverzeichnis kopieren oder bewegen (abhängig davon, ob du die Datei in dem Ursprungsverzeichnis auch behalten möchtest).

1. Finde das *plugins*-Verzeichnis in dem Hauptverzeichnis von Wireshark
2. In diesem, öffne das Unterverzeichnis, was nach der Versionsnummer von Wireshark benannt worden ist
3. Öffne eine neue Instanz von Windows Explorer mit dem Shortcut *Win + E*
4. Navigiere zu dem Dateispeicherort von der BAOS-Dissector-*.lua*-Datei
5. Kopiere oder schneide die *.lua*-Datei aus
6. Füge die *.lua*-Datei in das *"Hauptverzeichnis-von-Wireshark" / plugins / "Versionsnummer-von-Wireshark"* Verzeichnis ein

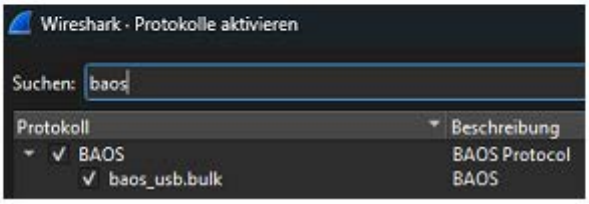
Wie wird Wireshark konfiguriert

Dies ist eine Schritt-für-Schritt-Anleitung für die Konfigurierung von Wireshark, damit das BAOS-Dissector-Plugin ordentlich genutzt wird.

Sicherstellen, dass das Plugin geladen wurde

1. Suche nach Wireshark, indem du die *Windows*-Taste drückst, und `wireshark` eingibst
2. Öffne Wireshark
3. Öffne das *Analyse -> Protokolle aktivieren...* Untermenü
4. Suche nach `baos`

Wenn du einen Treffer wie auf dem untenstehenden Bild hast,
du hast das Plugin erfolgreich installiert.

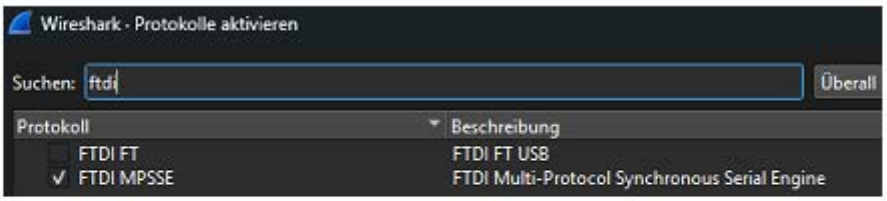


> Vergewissere dich bitte, dass das Kontrollkästchen neben dem Eintrag *baos_usb.bulk* angehakt ist!

Kollidierenden Dissector deaktivieren

Abhängig von der installierten Version von Wireshark, ein *FTDI FT*-Dissector ist wahrscheinlich aktiv. Da dieser mit dem BAOS-Dissector in Konflikt steht, ist es nötig, den *FTDI FT*-Dissector zu deaktivieren.

1. In dem gleichen *Protokolle aktivieren...* Untermenü wie in der vorherigen Sektion, suche nach `ftdi`
2. Entferne das Häkchen bei dem *FTDI FT*-Eintrag

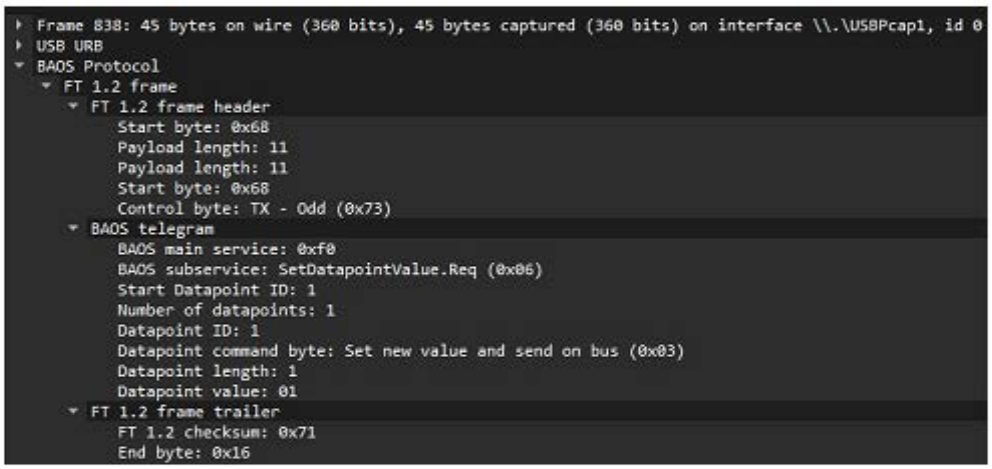


Wie wird der Dissector genutzt

Dies ist eine kurze Präsentation, wie der Dissector genutzt werden kann.

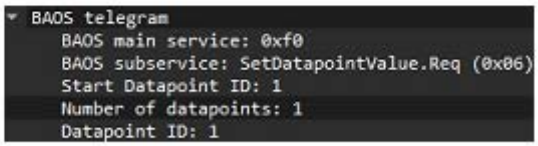
BAOS-Telegramme und -Felder erkennen und beschriften

Während der Überwachung einer Verbindung oder wenn eine abgespeicherte Mitschnittdatei geöffnet ist, wenn Wireshark, mit dem aktivierten BAOS-Dissector, ein BAOS-Telegramm erkennt, dieses Telegramm wird jetzt ordentlich als Paketbaum auf dem *Paketinformationen*-Panel dargestellt. Der FT 1.2-Frame-Header und der -Frame-Trailer um den BAOS-Payload herum werden auch zerlegt.



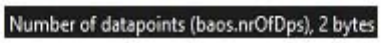
Der BAOS-Dissector integriert sich ordentlich in Wireshark. Das heißt, alle dazugehörigen Funktionalitäten werden unterstützt.

Wenn Protokollfelder auf dem *Paketinformationen*-Panel selektiert werden,



die dazugehörigen Hexadezimalwerte werden ebenso hervorgehoben,

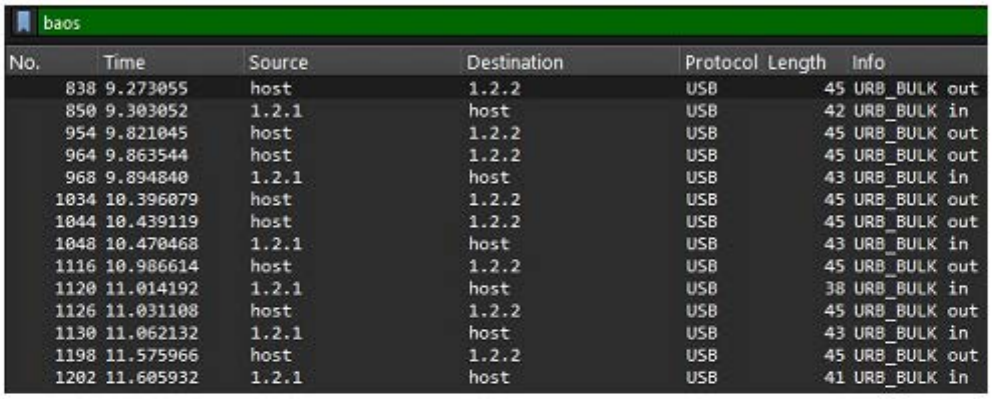
und der Feldname, der Filtername, die Wertgröße werden auf der Statusleiste angezeigt.



Filterung

Eine äußerst nützliche Funktion während der Überwachung von Verbindungen in Wireshark ist Filterung. Da der BAOS-Dissector sich in Wireshark vollständig integriert, wird die Filterung-Funktion ebenso unterstützt.

Indem du `baos` in das Suchfeld eingibst und *Enter* drückst, kannst du die überwachte Kommunikation in Wireshark filtern, wodurch nur die Pakete, die das BAOS-Protokoll nutzen, angezeigt werden.



Weitere, noch präziser Filterlösungen für gewisse Protokollfelder werden auch unterstützt. Nachdem du `baos` eingegeben hast, ergänze die Eingabe um einen `.` *Punkt*. Danach werden all die verfügbaren Filteroptionen für das BAOS-Protokoll von Wireshark vorgeschlagen.