Entwicklung und Implementierung eines Netzwerkprotokolls zur bidirektionalen Kommunikation und Steuerung unter Zuhilfenahme einer selbstentwickelten Sprache zur Eingabe von Befehlen

Jan Sommerfeld Leonard Petereit Benedikt Schäfer

19. November 2018

Inhaltsverzeichnis

1 Entwurf und Umsetzung der Netzwerk-Kommunikations-Schnittstelle

1.1 Grundlegende Theorie zur Kommunikation in einem Netzwerk

Unter dem Begriff Netzwerk im Zusammenhang mit dieser Arbeit wird, sofern es nicht ausdrücklich anders definiert ist, die Ansammlung mehrerer, untereinander verbundener Computer verstanden. Diese haben die Möglichkeit mit standardisierten Internet-Netzwerkprotokollen miteinander zu kommunizieren. Das umfasst das ISO/OSI Modell der Netzwerkkommunikation, auf dem auch die Arbeitsweise unseres Programmes aufbaut. Dabei kann man diesen Prozess und die verwendeten Informationen in Schichten einteilen, welche nach der Abstraktion von Nullen und Einsen in einem Kabel bis hin zu z.B. verschiedensten Verschlüsselungsprotokollen oder Websites eingeteilt werden; was in unteren Schichten abgesichert ist, muss in denen darüber nicht implementiert werden, was wiederum zu relativ einfachen einzelnen Algorithmen zur Kommunikation führt, obwohl das gesamte System überaus komplex ist.

1.2 Notwendigkeit, Anforderungen und Spezifikation eines eigenen Netzwerkprotokolls

Die Aufgabe der Bibliothek ist, eine verlässliche Verbindung zwischen zwei Rechnern aufzubauen(Start und Ziel), mit der es möglich ist, einen festen Satz von Anweisungen mit optionalen Argumenten sowie Dateien bidirektional zu übertragen. Dafür stellen sich folgende Anforderungen, es muss:

- ein gesicherter Transport von beliebigen Daten möglich sein,
- eine verschlüsselte Kommunikation vorliegen
- der Datenverkehr für Anweisungen so klein wie möglich sein
- der Kommunikationskanal in beide Richtungen gleich aufgebaut sein und
- die Übertragung auch von größeren Dateien mit zusätzlichen Informationen fehlerfrei ablaufen

Diese Bedingungen implizieren bereits, dass eine bestimmte Regelung und einen Ablauf für die Kommunikation zwischen den beiden Computern geben muss: ein Protokoll (s. ***Protokoll***).

Alle oben genannten Anforderungen müssen durch dieses Protokoll erfüllt und geregelt sein; daher umfasst es sowohl einen bestimmten Datensatz, der in den Header (s. ***Header***) eines Pakets (s. ***Paket***) geschrieben wird als auch einen festgelegten Ablauf der Kommunikation, der sicherstellt, dass alle Daten korrekt angekommen sind und verarbeitet wurden.

Um nicht ebenfalls für die sichere Ankunft der Daten sorgen zu müssen, baut das Protokoll auf TCP/IP auf (genauer auf dem Protokoll TLS, welches auch eine verschlüsselte Verbindung aufbaut), also der Transportschicht des ISO/OSI-Modells, auf der bereits genau das implementiert und standardisiert ist - praktisch ist unser Protokoll also auf der Anwendungsschicht des Modells. Da das Protokoll sowohl den Versand von Anweisungen als auch den von Dateien kontrollieren muss und dabei so klein wie möglich sein soll, gibt es zwei verschiedene Teilprotokolle, welcher in folgendem Aufbau der Kommunikationsstruktur resultiert:

Im ersten Schritt werden zwei Informationen zum Typ der Übertragung, also Anweisung oder Datei, und zu ihrer Gesamtgröße in den Header des ersten Pakets der Übertragung geschrieben. Abhängig davon, was der Übertragungstyp ist, wird nun entweder die Informationen für eine Dateiübertragung oder eine Anweisungsübertragung angehängt. Dabei sind alle Werte, sofern

es möglich ist, als Zahlen und nicht als Zeichenketten vorhanden, um die Größe der Informationen zu minimieren.

Betrachtet man zunächst den Header für die Anweisungen, so braucht man einen Anweisungscode, der am Zielort eindeutig zu einer Anweisung zugeordnet werden kann; Teil zwei sind Standardargumente, die binär auf 32 Bit Speicherbreite gespeichert werden. Dabei soll zum Beispiel, wenn das erste Bit auf eins gesetzt wurde, die Anweisung „Rufe eine Liste der Daten ab, auf die Das Programm Zugriff hat“ nur die Ordnerstruktur herüberschicken, ist das zweite Bit auf eins gesetzt soll es zusätzliche Informationen zur Größe der bzw. Zugriffsrechte auf die Dateien mitsenden; dafür wären anderenfalls eigene Anweisungen nötig, was der Übersichtlichkeit bei der Implementierung abträglich wäre.

Der dritte Wert im Headersegment für die Anweisungen ist eine Zahl, die ein bestimmtes Programm identifiziert, auf dass die Anweisung angewendet werden soll, was standardmäßig das Eigene ist. Der vierte Wert ist eine Längenangabe in Bytes. Er bezeichnet die Länge eines optionalen Argumentes, welches an den Header angehängt werden kann, beispielsweise ein Dateiname oder eine Chat-Nachricht.

Um die ganze Anweisung klein zu halten wird intern festgelegt, dass die Gesamtgröße dieses Headersegments inklusive des optionalen Arguments nicht größer als 2^{16} Byte sein darf. Aus Kompatibilitätsgründen mit verschiedenen Compilern und Systemen sind alle Werte Integer mit 32 bzw. 64 Bit Speicherbreite, um damit sowohl genügend Platz für alle Informationen zu bieten als auch Probleme mit verschiedenen Rechnerarchitektur-spezifischen Anpassungen der Speicherbreite zu vermeiden. Insgesamt ergibt sich also folgende Spezifikation des Headers für die Versendung von Anweisungen (Abb. 1):

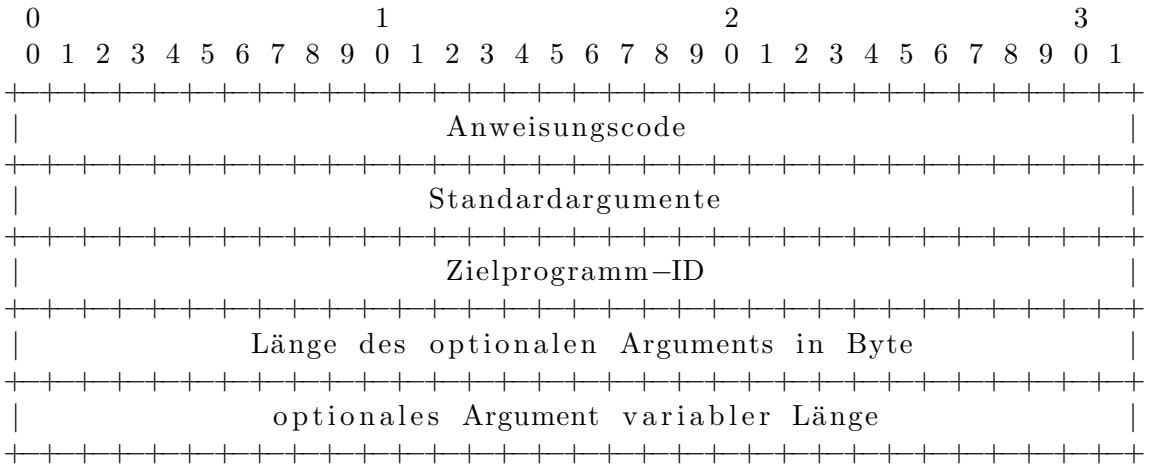

Abbildung 1: Header für Anweisungen

Dabei ist die Länge der Argumente in (pro Zeile 32) Bit angegeben, sofern nicht anders spezifiziert.

Das Versenden von Dateien hat andere Anforderungen, was in einem anderen Aufbau des Header-Segments für eine solche Übertragung, sowie in einem anderen Ablauf resultiert.

Für jede Dateiübertragung wird, im Gegensatz zu den Anweisungen, jeweils eine neue TLS-Verbindung aufgebaut(Schritt 0), die unabhängig von vorhergehenden und nachfolgenden ist, es gibt also für jede Datei einen abgeschotteten „Kanal“, in dem Fehler leicht behandelbar und konkurrierende Übertragungen unmöglich sind. Danach wird ein Paket mit Informationen zu der Datei zum Zielcomputer geschickt(Schritt 1), welcher, um die Integrität der Übertragung zu verifizieren, dasselbe Paket zurücksendet(Schritt 2). Dieses Informationspaket ist in Abbildung

2 schematisch dargestellt.
Es besteht aus der Art der Datei (z.B. Video, Audio, Text, Ausführbare Dateien etc.), was in der weiteren Einordnung im Zielcomputer nützlich ist, dem ursprünglichen Dateinamen und einer Prüfsumme, mit der die fehlerfreie Ankunft verifiziert werden kann.
Das Headersegment ist daher so aufgebaut, dass zunächst die drei Ganzzahlwerte für Typ, Länge des Dateinamen in Byte und Länge der Prüfsumme in Byte in den Header geschrieben werden. Dann werden Dateinamenslänge und Prüfsumme angehängt und dieses gesamte Paket wird an den Zielcomputer übermittelt.

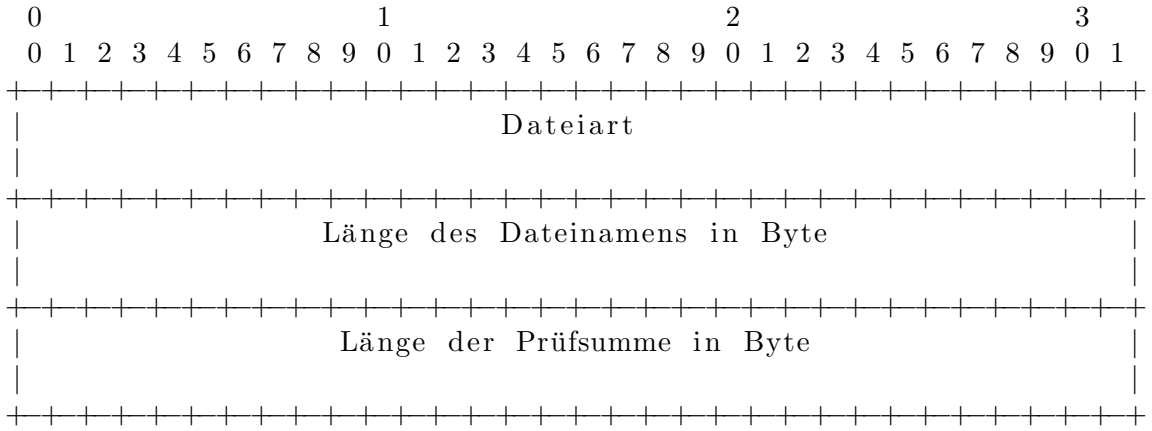

Abbildung 2: Header für Dateiübertragungen

Ist die Verbindung verifiziert, wird mit der eigentlichen Übertragung der Datei begonnen, die in kleinen Abschnitten und ohne weitere Kennzeichnung von Paketen geschieht(Schritt 3 bis x). Das funktioniert, da unser Standard auf TCP/IP aufbaut, welcher bereits die richtige Reihenfolge und Vollständigkeit der Pakete weitgehend sicherstellt und nur eine Verbindung pro Dateiübertragung aktiv ist. Am Zielcomputer wird die Datei schließlich in einem temporären Standardverzeichnis abgespeichert und zusammengesetzt. Ist die Übertragung nach dem Senden einer bestimmten Bytesequenz vom Zielcomputer beendet, wird ein Signal aus der Managerklasse ausgesendet und die Verbindung getrennt. Diese Schrittfolge ist schematisch in Abbildung 3 dargestellt.

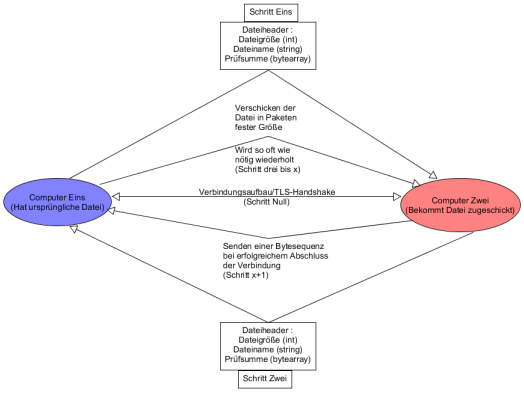

Abbildung 3: Schematische Darstellung des Ablaufs bei der Dateiübertragung

1.3 Funktionsumfang der selbstgeschriebenen Klassenbibliothek

Die Bibliothek stellt auf oberster Ebene eine Schnittstelle zum Übertragen von Dateien und Anweisungen zu einer bekannten Adresse, welche in angemessener Form codiert sind. Das heißt konkret, es werden Methoden bereitgestellt, welche die einzelnen Informationen in das passende Format das einfach verschickt werden kann umwandeln, dargestellt durch zwei Klassen, die die nötigen Informationen, also z.B. Dateiprüfsummen, Dateinamen, Anweisungen usw., so verpacken, dass diese dann einfach vom darunterliegenden System verarbeitet werden können. Zusätzlich dazu ist natürlich eine entsprechende Fehlerbehandlung bei Verbindungsabbrüchen

oder fehlerhaften Übertragungen und die Möglichkeit eine Fortschrittsmeldung bei der Dateiübertragung implementiert. Insgesamt muss der Programmierer nur mit zwei Klassen interagieren, um die volle Funktionalität der Bibliothek ausnutzen, was das gesamte Konstrukt sehr gut für weitere Verwendungen in flexiblen Kontexten der modularen Programmierung eignet und gute Modularität bietet.

1.4 Implementierung und Funktionsweise der Klassenbibliothek

In der Bibliothek sind neben den beiden oben beschriebenen Manager-Klassen mehrere andere Klassen definiert, welche von den jeweiligen Manager-Klassen aus aufgerufen und benutzt werden. Außerdem gibt es eine von allen Klassen benutzte Datei, in denen die Datenstrukturen und Datentypen, die in den Header der Pakete geschrieben werden, definiert sind und einige Funktionen, die häufig gebraucht werden (z.B. String-Vervielfältigung, eine Hexadezimale Ausgabe und eine Funktion zum errechnen der Dateiprüfsumme) deklariert; die Definition erfolgt in einer zugehörigen Datei.

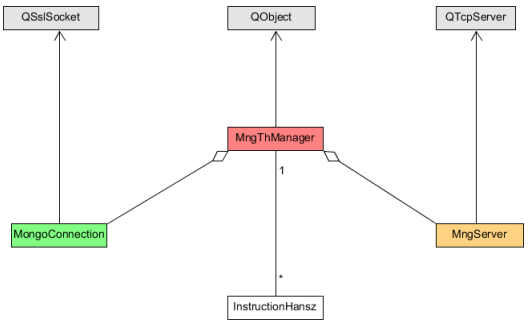

Abbildung 4: Schematisch: Anweisungsversand

Der Aufbau der Übertragung beider Datentypen, die verschickt werden, ähnelt sich bis zu einem gewissen Punkt - so haben beide eine eigene Klasse, in der intern die Objekte(Anweisungen und Dateizugriffsobjekte) gespeichert sind und verwaltet werden: (*FileHansz* bzw. *InstructionHansz*), sowie andere Klassen zum Verbindungsaufbau und dem Versand: Eine Klasse, die für das Annehmen von hereinkommenden Verbindungen zuständig ist und eine(bei Dateien zwei) Socket-Klassen, die sich um das Senden bzw. Empfangen von Daten kümmern, was in Abb. 4 bzw. 5 vereinfacht schematisch dargestellt ist.

Die beiden Speicherklassen werden dabei sowohl für den Empfang als auch das Senden eines Objektes genutzt - im ersten Fall werden sie mit den Teildaten(also Anweisungscode und zusätzlichen Argumenten oder Dateiname und Dateityp) initialisiert und erstellen daraus einen Pufferspeicher, der bereits den korrekten Header und das Datenpaket enthält, dessen Inhalt dann nur noch ausgelesen und verschickt werden muss.

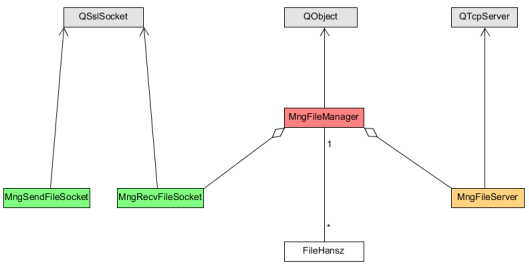

Abbildung 5: Schematisch: Dateiversand

Das tatsächliche Versenden, sowie der Verbindungsaufbau wird von einer „Server“-Klasse und Socket-Klassen übernommen. Dabei horcht die Serverklasse, sowohl für Dateien als auch für Anweisungen auf der Empfangenden Seite auf einem zuvor bestimmten Port auf eingehende Verbindungen. Auf der sendenden Seite wird eine neue Klasse erstellt, welche einen TLS-Socket initialisiert und zu der gegebenen Adresse verbindet. Ist die Verbindungsanfrage eingegangen, wird eine normale TLS-gesicherte Verbindung aufgebaut, sofern das möglich ist; danach weichen die Abläufe von Dateiversand und Anweisungsübertragung ab: Anweisungen werden jetzt, in der Reihenfolge in der sie eingehen, von der Managerklasse an das Socket getaktet weitergeleitet, wo sie dann inklusive Header und Datenpaket verschickt werden - bei Dateien ist der Ablauf etwas komplexer.

Dabei muss zunächst erwähnt werden, dass für Dateien, im Gegensatz zu Anweisungen jedes

Mal eine neue Verbindung aufgebaut wird. In der Verbindung für eine Datei wird, sobald sie aufgebaut ist, zuerst der Header für die Dateiübertragung an den Zielcomputer gesendet(1), welcher genau dasselbe Paket dann zurückschickt(2) um sicherzustellen, dass alle Steuerinformationen korrekt angekommen sind. Jetzt wird die eigentliche Übertragung der Datei gestartet, welche mit dem Auslesen eines Datenblockes fester Größe beginnt(3). Dieser wird in einen Pufferspeicher geschrieben, der, wenn er voll ist von der Socket-Klasse ausgelesen und übertragen wird(4); diese Anweisungsfolge(3 und 4) wird jetzt so lange wiederholt, bis das Ende der Datei erreicht ist. Der letzte Datenblock enthält meistens weniger als die Maximalgröße, was aber in der Praxis kein Problem ist - das letzte Datenpaket auf der Seite des verschickenden Rechners ist nur etwas kleiner als der Rest. Auf der empfangenden Seite werden die ankommenden Datenpakete über ihre Repräsentation als FileHansz-Objekt in eine nach der Datei-Prüfsumme benannten Datei im Dateisystem gespeichert - das verhindert die konkurrierende Benennung von ungleichen Dateien und vereinfacht das Überprüfen auf korrekte Übertragung; die ursprünglichen Namen werden zu Beginn der Übertragung per Signal an das Programm außerhalb der Bibliothek geschickt, welches diese dann weitergehend speichert. Sind alle Pakete versandt worden und angekommen, wofür die TCP-basierte Verbindung sorgt, wird vom Empfangenden Rechner eine Bytesequenz übertragen, welche die Erfolgreiche Übertragung kennzeichnet.

Danach wird die Verbindung abgeschlossen und die beiden Sockets gelöscht.

Anhang

Glossar

Header:	Ein Datensatz der an den Anfang eines im Netzwerk verschickten Paketes geschrieben wird um dessen sichere Ankunft sicherzustellen oder ein Teil des Quelltextes, in dem bestimmte Datentypen und Strukturen definiert aber nicht deklariert ist.
Paket, Netzwerkpaket:	Informationseinheit, die von Computer zu Computer in einem Netzwerk versendet wird
Protokoll, Netzwerkprotokoll:	Spezifikation eines Teils des Headers eines Pakets; kann außerdem auch einen Ablauf für die Kommunikation zwischen den an der Verbindung beteiligten Rechnern beinhalten
ISO/OSI Modell	Ein informatisches Modell zur Darstellung der Kommunikation in einem Netzwerk. Dabei werden verschiedene „Schichten“ definiert, die jeweils unterschiedliche Aufgaben bei der Kommunikation übernehmen und aufeinander aufbauen. So gibt es die unteren, noch sehr hardwarenahen Schichten, die sich um die korrekte Übertragung von einzelnen Bits kümmern; Schichten darüber, welche für die Ankunft einzelner ***Paket***e am richtigen Ort zuständig sind bis hin zu Schichten, die Abläufe und Formate enthalten, die für die Darstellung und Übertragung von Websites zuständig sind.