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

26. Oktober 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 nach Internet-Standards miteinander zu kommunizieren. Das umfasst das ISO/OSI Modell der Netzwerkkommunikation, auf dem auch die Arbeitsweise unseres Programmes aufbaut. Kommunikation in so einem Netzwerk basiert auf dem verschicken von kleinen Informationseinheiten, genannt *Paket*, welche mit verschiedenen Steuerungsinformationen versehen werden, welche vor das Paket gehängt werden, in einem Segment, das *Header* heißt, um an ihr Ziel zu gelangen. Zu diesem Zweck besitzt jeder Computer bestimmte Adressen, die in den Header des zu verschickenden Pakets am Ausgangspunkt geschrieben werden, um am Ziel anzukommen. 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

Unser Programm soll am Ende einen Kommunikationskanal zwischen einem Start- und einem Zielrechner aufbauen. Zwischen den Rechnern sollen bestimmte Anweisungen, z. B. zum Speichern oder Abrufen von Dateien oder dem Ausführen bestimmter Programme hin und her gesendet werden können; außerdem soll dasselbe mit beliebigen Dateien direkt möglich sein. Um dieses Ziel zu erreichen 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 flexibel 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.

Alle oben genannten Anforderungen müssen durch dieses Protokoll erfüllt und geregelt sein; daher umfasst es sowohl einen bestimmten Datensatz, der in den Header eines Paketes 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 dieses Protokoll also auf der Anwendungsschicht des Modells. Da das Protokoll sowohl den Versand von Anweisungen als auch den Versand von Dateien

kontrollieren muss und dabei so klein wie möglich sein soll, gibt es im Grunde zwei verschiedene Protokolle. Trotzdem existieren einige gemeinsame Informationen, namentlich Übertragungstyp und Gesamtgröße, die übertragen werden müssen; so ergibt sich ein dreiteiliger Aufbau der Kommunikation:

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 dahinter geschrieben. Dabei sind alle Werte als Zahlen nicht als Zeichenketten vorhanden, um die Größe zu optimieren.

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. Da Anweisungen immer einem Programm zugeordnet werden, für das sie ausgeführt werden(das Eigene eingeschlossen) wird diese ID benötigt. Sie wird von der Netzwerkanbindung aber nicht weiter behandelt, sondern erst im zweiten Programmteil tatsächlich benutzt um eine bestimmte Aktion zu steuern. 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 darf die Gesamtgröße dieses Headersegments inklusive des optionalen Arguments nicht größer als 2^{16} Byte sein. Aus Kompatibilitätsgründen mit verschiedenen Compilern und Systemen sind alle Werte Integer mit 32 bzw. 64 Bit Speicherbreite. Insgesamt ergibt sich also folgende Spezifikation für die Versendung von Anweisungen(Abb. 1 und 2):

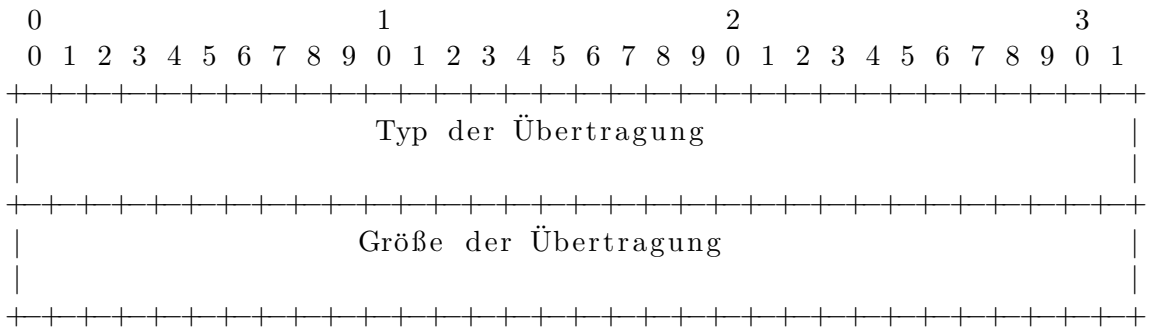

Abbildung 1: Allgemeiner Header

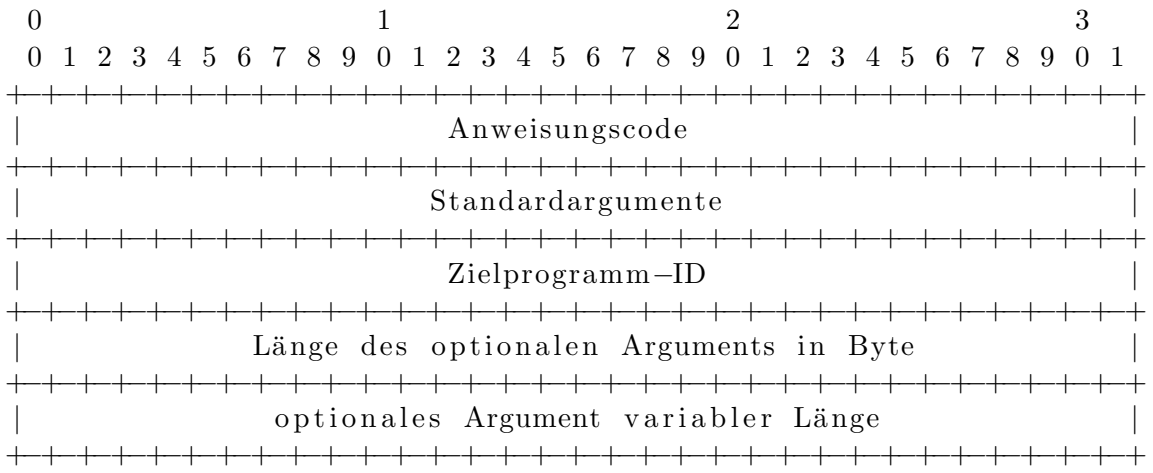

Abbildung 2: Header für Anweisungen

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 Dateiübertragung resultiert.

Für jede Dateiübertragung wird, im Gegensatz zu den Anweisungen, jeweils eine neue TCP/IP-Verbindung aufgebaut, die unabhängig von vorhergehenden und nachfolgenden ist. Danach wird ein Paket mit Informationen zu der Datei zum Zielcomputer geschickt, welcher um die Integrität der Übertragung zu verifizieren dasselbe Paket zurücksendet. Dieses Informationspaket sieht so aus:
Es ist nötig den Dateityp zu kennen, um die Datei später einordnen zu können; dieser wird wieder als Ganzzahl gespeichert. Weiterhin ist der Dateiname wichtig, ebenso wie eine Datei-Prüfsumme, mit der bestätigt werden kann, dass die Datei vollständig und korrekt angekommen ist.
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 angehangen und dieses Paket wird an den Zielcomputer übermittelt.

Die Spezifikation sieht also so aus(Abb.1 und 3)

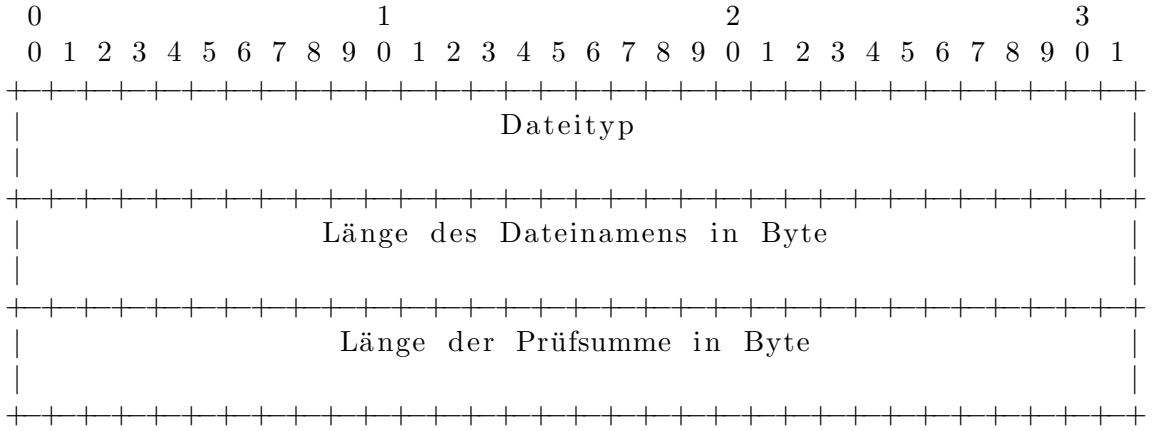

Abbildung 3: Header für Dateiübertragungen

Ist die Verbindung verifiziert, wird mit der eigentlichen Übertragung der Datei begonnen, die in kleinen Abschnitten und ohne weitere Kennzeichnung versendet wird. Das funktioniert, da unser Standard auf TCP/IP aufbaut, welche bereits für die korrekte Ankunft innerhalb einer Verbindung sorgen. Am Zielcomputer wird die Datei schließlich in einem temporären Standardverzeichnis abgespeichert und zusammengesetzt. Ist die Übertragung beendet, wird ein Signal ausgesendet und die Verbindung getrennt.

1.3 Funktionsumfang der selbstgeschriebenen Klassenbibliothek

Die Implementierung des zuvor beschriebenen Protokollstandards stellt auf der obersten Ebene zwei Hauptklassen bereit; eine für Anwendungen und die andere für Dateien. Außerdem existieren zwei Klassen um Dateien und Anweisungen zu speichern und verwalten: FileHansz für Dateien und InstructionHansz für Anweisungen. Beide arbeiten so, dass zunächst eine Verbindung spezifiziert werden muss mit einer IP beliebiger Version und (optional) einem TCP-Port. Nach der Initialisierung durch die IP bzw. die Ports wird eine Verbindung zu einem entsprechenden Gegenstück aufgebaut und bei Erfolg bzw. Misserfolg ein Signal ausgesendet, das von dem Anwendungsprogrammierer behandelt werden kann. Steht die Verbindung kann man relativ unkompliziert entweder eine der Speicherklassen oder die Informationen einer Anweisungen bzw. ein Dateiobjekt übergeben, das an das entsprechende Ziel gesendet werden soll, was dann auch ausgeführt wird, ohne weitere Interaktion oder Einstellungen von außen zu erhalten. Bei bestimmten Ereignissen(Verbindung aufgebaut, Verbindung getrennt, Dateiübertragung begonnen, Dateiübertragung beendet etc.) werden wieder Signale ausgesendet, die im Falle einer erfolgreich abgeschlossenen Übertragung auch einen Zeiger auf das entsprechende Speicherobjekt hat, welches dann weiter im Programm verwendet werden kann. Außerdem wird, im Falle einer Datei, in regelmäßigen Abständen die hereingekommenen bzw. ausgesendeten Daten in Bytes an die anderen Programmteile weitergegeben um einen Fortschritt bestimmen zu können. Damit bietet die Bibliothek eine gute Modularität und eine einfache Anbindung an weiter Programme, was das Projekt an sich vereinfacht.

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 einer Dateiprüfsumme) deklariert; die Definition erfolgt in einer zugehörigen Datei.

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 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.

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.

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 bestimmten Port auf Verbindungen. Auf der sendenden Seite wird eine neue Klasse erstellt, welche einen Web-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, je nachdem wie 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 vom Socket 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 Dateien und vereinfacht das Überprüfen auf korrekte Übertragung; die ursprünglichen Namen werden zu Beginn der Übertragung an das weitere Programm 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.