rdrop - Projekt Dokumentation

Lars Zocher, Simon Stiegler, Philipp Eichwald

14. Mai 2023

1 Einleitung

In der modernen Welt ist der Austausch von Daten ein immer wichtiger werdender Bestandteil des alltäglichen Lebens. Dennoch bieten die meisten Anbieter für das Übertragen von Daten keine Möglichkeit, die Daten auf direktem Wege zwischen den Nutzern auszutauschen. In den meisten Fällen müssen die Daten zunächst auf einem Server zwischengespeichert werden, damit die empfangende Partei diese Daten herunterladen kann.

Dieser Umweg ist nicht nur Zeitaufwändig, sondern auch ein Sicherheitsrisiko, da die Anbieter, die die Daten zwischenspeichern, Zugriff auf die Daten erhalten [1]. Eine Alternative dazu bietet AirDrop. AirDorp ist ein Dienst der Firma Apple der es ermöglicht Daten direkt zwischen zwei Geräten zu übertragen. Jedoch besitzt AirDrop den entscheidenden Nachteil, dass sich die Geräte in unmittelbarer Nähe zueinander befinden müssen [2].

Alle bisherigen Lösungen benötigen mindestens einen Server, um die Kommunikation zwischen beiden Parteien aufzubauen. Eine wirklich einfache Lösung für die gesicherte direkte Übertragung von Daten zwischen zwei Parteien exisitiert bisher nicht.

Ziel dieses Projekts ist es, eben dieses Problem zu lösen. Dafür wird eine Anwendung entwickelt, die es ermöglicht ohne die Verwendung eines Server Daten auf direktem Wege zwischen zwei Parteien auszutauschen. Die Anwendung soll dabei auf allen

gängigen Betriebssystemen lauffähig sein. Insbesondere wird ein Fokus auf die unkomplizierte Bedienung der Anwendung gelegt, um so auch unerfahrenen Nutzern die Möglichkeit zu bieten, Daten sicher und schnell auszutauschen.

Das folgende Dokument beinhaltet die Dokumentation des Projekts. Dabei werden zunächst die Anforderungen an die Anwendung definiert. Anschließend wird die Architektur der Anwendung erklärt und wie diese umgesetzt wurde. Schlussendlich wird die Umsetzung und Lösung des Projekts kritisch reflektiert und ein Ausblick auf mögliche Erweiterungen gegeben.

2 Anforderungen

Im folgenden sind die Anforderungen an die Software definiert. Die Anforderungen werden dabei in funktionale und nicht funktionale Anforderungen unterteilt.

Funktionale Anforderungen

Anforderung 1: Lauffähigkeit Die Software soll auf allen gängigen Betriebssystemen lauffähig sein.

Anforderung 2: Übertragung Die Software soll es ermöglichen, ohne die Verwendung eines Servers, Daten direkt zwischen zwei Parteien zu übertragen.

Anforderung 3: Sicherheit Die gesamte Kommunikation zwischen den Parteien soll verschlüsselt erfolgen.

Anforderung 4: Dateien Die Software soll die Möglichkeit bieten, Dateien zwischen den Parteien auszutauschen. Diese sollen automatisch aus dem eingelesen und abgespeichert werden.

Anforderung 5: Programmiersprache Die Software muss mit der Programmiersprache Rust umgesetzt werden.

Anforderung 6: Zuverlässigkeit Die Kommunikation zwischen den Parteien muss auf eine zuverlässige Art und Weise erfolgen. Die Pakete müssen reihenfolgengetreu und verlustfrei empfangen werden können.

Nicht-Funktionale Anforderungen

Anforderung 1: Benutzerfreundlichkeit Die Software soll eine leicht verständliche Oberfläche bieten. Zu jedem Zeitpunkt muss klar seine welche Aktionen möglich sind. Es werden keine überflüssigen Informationen angezeigt.

3 Architektur

Insgesamt ist die Softwarelösung in die drei Module Benutzeroberfläche, Kommunikation und Datei-Ein-/Ausgabe aufgeteilt. Grund dafür ist, dass insbesondere die Module Kommunikation und Datei-Ein-/Ausgabe auch unabhängig voneinander arbeiten können.

So wird die Kohäsion der einzelnen Teile erhöht und die Kopplung verringert. Im foldenden werden die einzelnen Teile genauer erklärt.

3.1 Benutzeroberfläche

architektur und so

3.2 Kommunikation

Das Modul Kommunikation ist für den Verbindungsaufbau und den Nachrichtenaustausch zwischen den Parteien zuständig. Nach außen soll es eine stark abstrahierte Schnittstelle bieten, die Fehlbenutzung vermeidet und die Verwendung maximal vereinfacht.

Das Modul soll in die Untermodule *Client* und *Protokoll* aufgeteilt werden. Das Modul *Client* bietet dabei die Schnittstellen für das Senden und Empfangen von Nachrichten. Mithilfe des *Protokoll* Moduls wird die Möglichkeit geboten die Verbindung zweier *Client* Instanzen aufzubauen und zu verschlüsseln.

Insgesamt sollen drei Umsetzungen eines Clients implementiert werden. Darunter fallen zwei Basisimplementierungen, die die Kommunikation über TCP und UDP ermöglichen. Die dritte Implementation baut auf einer der Basisimplementierungen

auf und verschlüsselt die entsprechende Kommunikation.

Für jeden Client besteht jeweils die Möglichkeit den Sender- und Empfängerteil in zwei seperate Objekte aufzuteilen. Somit wird sichergestellt, dass Anwender die Kommunikation auf mehrere Threads aufteilen können.

Das Protokoll verwendet die implementierten Clients und sorgt für eine reibungslosen Übergangt zwischen den verschiedenen Implementationen. Dies ist notwendig, da zunächst nur eine UDP-Kommunikation aufgebaut werden kann, da das TCP-Protokoll eine zeitliche Synchronisation benötigt. Diese Synchronisation kann nur über eine bereits existierende Verbindung erfolgen.

Um die Verbindung während des Aufbaus der TCP-Verbindung abzusichern ist der erste Schritt nachdem UDP-Verbindungsaufbau die Verschlüsselung der Verbindung. Anschließend erfolgt ein Versuch die Zeiten zu synchronisieren und eine TCP-Verbindung aufzubauen. Bei Erfolg wird die Kommunikation in eine verschlüsselte TCP-Verbindung überführt, andernfalls wird die bereits bestehende UDP-Verbindung weiter verwendet.

Benutzern der Schnittstelle ist offengestellt welche Art der Kommunikation sie verwenden möchten. Das Protokoll kann zu jedem Zeitpunkt in einen aktiven Client überführt werden, über den dann entsprechnder Datenaustausch möglich ist.

3.3 Datei-Ein-/Ausgabe

architektur und so

4 Umsetzung

Dieses Kapitel beschreibt die Umsetzung der in Kapitel **??** beschriebenen Architektur. Dabei wird auf die einzelnen Module eingegangen und die Umsetzung erläutert.

Die Module Kommunikation und Datei-Ein-/Ausgabe werden dabei als Bibliotheken implementiert, die von der Benutzeroberfläche verwendet werden. Die Benutzeroberfläche wird als eigenständiges Programm implementiert.

4.1 Benutzeroberfläche

umsetzung und so

4.2 Kommunikation

Für die Umsetzung der Clients werden die notwendigen Funktionalitäten, die an einen Client gestellt werden, in Traits festgelegt. Jeder der Clients implementiert diese. Somit können alle der Clients im generischen Kontext austauschbar verwendet werden.

Die Implementierung des TCP-Clients stellt dabei keine besondere herausforderung dar. In Addition zu dem standardmäßigen Protokoll muss lediglich die Länge eines Pakets hinzugefügt werden. Dies ist notwendig, da in einem TCP-Stream keine Paketgrenzen vorhanden sind.

Nach der 6. Funktionalen Anforderung muss die Kommunikation zuverlässig erfolgen. Aus diesem Grund muss für den UDP-Client ein Protokoll implementiert werden, das die Zuverlässigkeit gewährleistet. Da der UDP-Client im Kontext der Anwendung nur für den Kommunikationsaufbau verwendet werden soll, wird das Send-and-Wait Protokoll implementiert. Dieses Protokoll zeichnet sich durch seine Simplizität in der Implementierung aus. [3, S. 195]

Für die Implementierung wird ein seperater Thread erstellt, der alle ankommenden Pakete bestätigt und für den Anwender über einen Kanal zur Verfügung stellt. Ebenfalls ist ein Mechanismus implemntiert, der die Verbindung nach einer bestimmten Zeit abbricht, falls keine Antwort vom der Gegenseite empfangen wird.

Da die Kommunikation ebenfalls über das Internet läuft muss sichergestellt werden, dass die Firewalls durch welche die Pakete geroutet werden diese nicht verwerfen. Dazu werden Keep-Alive Pakete versendet, die die Löcher in den Firewalls geöffnet halten. [4]

Die Verschlüsselung der Kommunikation wird mithilfe der Bibliothek dryoc umgesetzt. Grund dafür ist, dass eine Implementierung einer eigenen Verschlüsselungsmethode ein besonders hohes Risiko bietet. Die Bibliothek dryoc bietet dabei eine simple

Schnittstelle, die die Verschlüsselung und Entschlüsselung von Daten ermöglicht.

Es wurde sich expliziert gegen die Verwendung einer zertifizierten Verschlüsselung entscheiden, da die Validität der Zertifikate nicht überprüft werden kann. Stattdessen kann die Kommunikation auch über einen entsprechenden Hash der Schlüssel validiert werden, der über einen seperaten Kanal ausgetauscht werden muss.

Eine Implementierung dieser Funktionalität ist bisher nicht implementiert, wäre aber eine sinnvolle Erweiterung, um Man-in-the-Middle Angriffe zu verhindern.

Die für die Kommunikation verwendete Verschlüsselungsmethode basiert auf ChaCha20-Poly1305. Diese Methode ist in der Lage, die Daten zu verschlüsseln und zu authentifizieren. [5]

Das Protokoll für den Verbindungsaufbau wird mithilfe eines Type-State-Patterns umgesetzt. Das Type-State-Pattern bietet den Vorteil, dass in jedem Zustand nur die erlaubten Aktionen ausgeführt werden können [6].

Der erste Status des Protokolls ist eine wartendes UDP-Sockets, dessen Port angefragt werden kann. Sobald beide Parteien ihre Ports ausgetauscht haben, kann ein Verbindungsaufbau versuch gestartet werden. Dafür sendet der Client in einem gegebenen Intervall Nachrichten an die andere Partei. Sobald eine Antwort empfangen wird ist die Verbindung erfolgreich. Andernfalls kann der Verbindungsversuch nach einer gegebenen Zeit automatisch abgebrochen werden.

Im nächsten Schritt werden die Rollen der Parteien festgelegt, da diese für die Verschlüsselung der Kommunikation benötigt werden. Dazu generiert jeder Client eine zufällige Zahl und tauscht diese mit dem Gegenüber aus. Sind beide Zahlen gleich, wird der Prozess wiederholt.

Anschließend werden die öffentlichen Schlüssel ausgetaucht und die Verschlüsselungsstreams initialisiert.

Falls gewünscht kann nun versucht werden die Kommunikation auf eine TCP-Verbindung umzutellen. Dafür müssen beide Parteien zeitlich synchronisiert werden, da beide TCP-Pakete gleichzeitig abgesendet werden müssen. Das implementierte Protokoll

verwendet dazu zwei Ansätze.

Zum einen werden Zeitstempel zwischen den Partein versendet und mit dem Rountrip-Delay verrechnet. Dies wird über mehrere Iterationen wiederholt, da aufgrund von asymmetrischem Netzwerkjitter niemals eine perfekte Synchronisation erreicht werden kann. Schlussendlich wird der Median der gesammelten Werte verwendet. Es gilt festzuhalten, dass falls der Hinweg dauerhaft länger als der Rückweg ist, keine Synchronisation möglich ist.

Alternativ besteht die Möglichkeit über einen externen NTP-Server die Zeit zu synchronisieren.

Sind beide Uhren vermeintlich synchronisiert stimmen beide Clients einen Zeitpunkt ab an dem das TCP-Paket abgesendet wird. Ist der Verbindungsaufbau erfolgreich, wird die UDP-Verbindung geschlossen. Andernfalls kann die UDP-Verbindung weiterhin verwendet werden.

4.3 Datei-Ein-/Ausgabe

umsetzung und so

5 Reflexion

Insgesamt konnten fast alle der gestellten Anforderungen im Projektzeitraum erfüllt werden. Ausgenommen ist die dritte funktionale Anforderungen, da die Software bisher keine Authentisierungmethode bietet.

5.1 Ablauf

Der Ablauf des Entwicklung kann als stolpernd beschrieben werden. Problematisch war insbesondere die fehlende Erfahrung mit Netzwerkkommunikation, der verwendeten Programmiersprache Rust und Multithreading. Das fehlende Wissen wurde erst mit dem Verlauf des Projekts erworben, wodurch viele früh implementierte Lösungen verworfen werden mussten.

Ursprünglich war nicht geplant eine Kommunikation über das UDP-Protokoll zu unterstützen. Nachdem sich der Verbindungsaufbau über TCP allerdings als eindeutig zu unverlässig erwieß, blieb keine andere Wahl.

Ebenfalls war zu Beginn des Projekts kein Verständnis für Netzwerkadressübersetzung vorhanden. Dies stellte sich als Problem heraus, weil dardurch keine direkte Kommunikation zwischen zwei Clients hinter seperaten Rechnernetzen zuverlässig aufgebaut werden kann. Somit musste im Verlauf des Projekts die Kommunikation mithilfe von IPv4 Adressen verworfen werden. Da Ipv6 keine Netzwerkadressübersetzung verwendet, war somit jedoch eine Alternative geboten.

Ein weiterer nicht miteinberechneter Faktor war der asymmetrische Netzwerkjitter. Dieser tritt insbesondere bei der Verwendung von Mobilen Daten auf und sorgt dafür, dass keine TCP-Verbindung hergestellt werden kann. Alternativ wird in diesem die UDP-Verbindung für den Dateiaustausch verwendet. Aufgrund der spezifischen Implementierung des UDP-Clients ist die Übertragungsgeschwindigkeit allerdings stark eingeschränkt.

Trifft man die Annahme, dass eine Verzögerung von 50 Millisekunden für den Weg von dem Sender zum Empfänger bei einer UDP-Paketgröße von einem Kilobyte besteht, so ergibt dies eine maximale Übertragungsgeschwindigkeit von maximal 10 Kilobyte pro Sekunde.

5.2 Ausblick

Die Softwarelösung bietet eine gute Grundlage für eine sichere und zuverlässige Kommunikation zwischen zwei Parteien.

Literatur

[1] Bundesamt für Sicherheit in der Informationstechnik, *Cloud: Risiken und Sicherheitstipps*. Adresse: `https://www.bsi.bund.de/DE/Themen/Verbraucherinnen-und-Verbraucher/Informationen-und-Empfehlungen/Cloud-Computing-`

Sicherheitstipps/Cloud-Risiken-und-Sicherheitstipps/cloud-risiken-und-sicherheitstipps_node.html.

[2] Apple Inc., *So verwendest du AirDrop auf einem iPhone oder iPad*. Adresse: https://support.apple.com/de-de/HT204144.

[3] Eidgenössische Technische Hochschule Zürich, *Vernetzte Systeme*, 2000. Adresse: http://www.vs.inf.ethz.ch/edu/WS0001/VS/Vorl.VNetz00_07.pdf.

[4] B. Seymour, *How To Create UDP Peer-To-Peer Connections With Netcat*, Juli 2021. Adresse: https://www.youtube.com/watch?v=TiMeoQt3K4g.

[5] Google, Inc., *ChaCha20 and Poly1305 for IETF Protocols*, Mai 2015. Adresse: https://tools.ietf.org/html/rfc7539.

[6] R. L. Apodaca, *Using the Typestate Pattern with Rust Traits*, Feb. 2023. Adresse: https://depth-first.com/articles/2023/02/28/using-the-typestate-pattern-with-rust-traits/.