Lan Messenger

Datentransfer in internen Netzwerken und sich daraus ergebende Probleme und Lösungen

Marek Küthe

Zusammenfassung
In dieser Arbeit beschäftige ich mich mit Filesharing in internen Netzwerken. Dazu werde ich bereits existierende Lösungen wie Snapdrop oder AirDrop betrachten.
Dabei konnte ich feststellen, dass diesen Lösungen noch Features fehlen. Ich habe mich daher entschlossen, ein eigenes Programm zu erstellen, welches die fehlenden Features implementiert. Dieses Programm werde ich in der Arbeit vorstellen und danach bewerten, in wie fern mein Programm die Features umgesetzt hat. Das fertige Programm kann man unter github.com/marek22k/lanmessenger finden.

Inhaltsverzeichnis

Problem
Wer kennt es nicht: Man möchte eine Datei von Gerät A nach Gerät B versenden, aber es gibt weder ein Programm, welches das andere Gerät findet, noch die Übertragung verschlüsselt?
Genau vor diesem Problem stand ich. Es sind zwar allgemeine Lösungen bekannt, diese sind aber herstellerbezogen und nicht-frei. Mit Herstellerbezogen meine ich Lösungen wie Nearby Share von Google oder AirDrop von Apple. Es ist unmöglich, Nearby Share mit einem iPhone oder iPad zu verwenden oder AirDrop mit Android. Dies kann sehr nervig sein.
Mit frei meine ich „‘Frei wie in Redefreiheit, nicht wie in Freibier’“ [download.fsfe.org/advocacy/promomaterial/FYA/Fdroid-leaflet/fdroid-leaflet.de.pdf, 2016, S. 1]. Es bezieht sich auf freie Software. Um es einfach auszudrücken ist freie Software wie Open-Source-Software, nur mit einer anderen Intention. Sie möchte dem Benutzer die Freiheit geben, über die Software zu bestimmen und vermeiden, dass die Software über den Benutzer bestimmt.

Hinweis
Ich möchte darauf hinweisen, dass meine Aussagen, welche ich im weiteren Text machen werden nur für den Fall gelten, wenn die Geräte im Netzwerk miteinander kommunizieren können bzw. dürfen. Dies wäre beispielsweise nicht der Fall, wenn man mit dem Gastzugang der Fritz!Box verbunden ist.

Kriterien
Um zu beurteilen, ob ich das Problem lösen konnte bzw. ob andere Lösungen es lösen konnte, habe ich einige Kriterien formuliert.

freie Software
Bei freier Software kann den Quellcode ansehen. Dadurch kann mit die Versprechen, welches ein Programm macht, verifizieren. Wenn beispielsweise ein Programm mit dem Versprechen der „Verschlüsselung“ bewirbt wird, kann man in den Quellcode schauen und dies verifizieren. Des Weiteren hat freie Software den Vorteil, dass andere Personen Sicherheitslücken finden können. Hier möchte ich auf das Sprichwort „Vier Augen sehen mehr als zwei“ verweisen. Wobei es bei freier Software auch mehrere Tausend Augen sein können.

Verbindung
Ein weiteres Kriterium wäre, dass sowohl der Verbindungsaufbau leicht ist als auch die nachfolgende Verbindung stabil ist. Wie ich weiter unten ansprechen werden, hat Snapdrop beispielsweise ein Programm damit, andere Geräte aufzuspüren.

Darüber hinaus sollte ich Verbindung zwischen den zwei Geräten stabil sein. Damit meine ich besonders Verbindungsabbrüche, welche nicht auf das Netzwerk, sondern auf das Programm zurückzuführen sind. Beispielsweise könnte irgendein „Cleaner“ im Hintergrund das Programm stoppen, obwohl die Datei noch nicht vollständig übertragen ist.

Integrität und Verschlüsselung
Außerdem sollte der Inhalt während des Transportes weder von Dritten gelesen oder verändert werden. Snapdrop beispielsweise verwendet WebRTC, welches standardmäßig verschlüsselt ist. Mein Programm verwendet OpenSSL für die Verschlüsselung und Integrität der Daten.

Plattformunabhängigkeit
Damit ein Programm für den Datenaustausch nützlich ist, sollte man es auf nahezu allen Geräten verwenden können. Die meist verwandtesten System sind wohl Android, Windows und i-Produkte.

Darüber hinaus wird manchmal auch noch Linux verwendet. Beispielsweise werde ich unten die Kompatibilität von AirDrop und Nearby Share beleuchten.

Es gibt verschiedene Möglichkeiten Plattformunabhängigkeit zu erreichen. Ich tue dies, indem ich seit Jahren existierende Bibliotheken sowie die Programmiersprache Ruby verwende.

Mehrere Dateien senden

Eine weitere praktische Fähigkeit für ein Filesharing-Programm, wäre mehrere Dateien auf einmal zu senden. Beispielsweise nimmt man im Urlaub häufig mehrere hundert Fotos auf und möchte diesen seinen Freund*in schicken. Jedes Foto einzeln auszuwählen, wäre sehr umständlich und nicht effizient. Daher sollte ein Filesharing-Programm die Fähigkeit besitzen, mehrere Dateien auf einmal senden zu können.

Existierende Lösungen

Apple’s AirDrop und Google’s Nearby Share

“AirDrop is a proprietary ad hoc service in Apple Inc.'s iOS and macOS operating systems [...]” (englisch: “AirDrop ist ein proprietärer Ad-hoc-Dienst in den Betriebssystemen iOS und macOS von Apple Inc [...]”) [en.wikipedia.org/wiki/AirDrop]. AirDrop ist also nicht frei. Es befindet sich im Eigentum von Apple.

Nearby Share wurde zwar von Google mitentwickelt und ist zwar auch Open-Source (www.reddit.com/r/chromeos/comments/ho83x8/google_is_making_nearby_sharing_opensource_code/, www.chromestsnaory.com/2020/07/nearby-sharing-edge/), dennoch findet man nach einer einfachen Recherche im Internet kein Repository, welches den Quellcode für das gesamte Nearby Share, wie es auch auf Androidgeräten verfügbar ist, enthält. Im Normalfall ist es so, dass freie Projekte auf ihrer Homepage einen Link zu GitHub oder einer ähnlichen Plattform hinzufügen, welcher dann auf den Quellcode des Projektes verweist.

Snapdrop

Dann kann man noch „normal“ im Internet nach anderen Lösungen schauen. Hier stößt man schnell auf das Programm Snapdrop (snapdrop.net). Dieses steht unter der GNU GPL v3 (github.com/RobinLinus/snapdrop/blob/master/LICENSE) zur Verfügung, welche eine freie Lizenz ist. Des Weiteren ist Snapdrop auf nahezu jedem internetfähigen Gerät verfügbar, da man es über eine Internetseite benutzten kann. Bei Bedarf gibt es auch Programme, welche man installieren kann. Darüber hinaus verfügt Snapdrop die Fähigkeit mehrere Dateien auf einmal zu senden. Je nach Einstellung werden alle Dateien angenommen oder man wird bei jeder Datei einzeln gefragt. Die Daten, welche über Snapdrop transportiert werden, sind zwar verschlüsselt (github.com/RobinLinus/snapdrop/blob/master/docs/faq.md#what-about-security-are-my-files-encrypted-while-being-sent-between-the-computers), dennoch habe ich bei Snapdrop ein Probleme erkannt. Dieses besteht darin, dass Snapdrop manchmal Schwierigkeiten bei der Findung des anderen Gerätes hat. Dies kann unter anderem daran liegen, dass Sicherheitsmaßnahmen auf den Geräten die Verbindung stören wie zum Beispiel VPN oder Firewall.

Dieses Problem möchte ich lösen, indem ich eine manuelle Eingabe der IP-Adresse des anderen Gerätes anfordere. So wird das andere Gerät direkt kontaktiert ohne den Versuch, es vorher aufzuspüren. Um die Sicherheit der Übertragung zu gewährleisten, ist es bei meinem Programm nötig, eine manuelle Überprüfung der Prüfsumme, also das Vergleichen von zwei Zeichenfolgen, durchzuführen. Dadurch wird festgestellt, ob sich das Gerät sich auch mit dem richtigen verbunden hat.

Sharik

Des Weiteren gibt es das Programm Sharik. Mit diesem kann man Dateien mit einem oder mehreren Netzwerkteilnehmer teilen. Dieses setzt dafür einen HTTP-Server auf. Ruft man nun die Seite auf, welche von HTTP-Server betrieben wird, kann man die gesendete Datei empfangen. Die Verbindung wird dabei nicht verschlüsselt. Des Weiteren braucht man keinen Nachweis, dass man berechtigt ist, die Datei zu empfangen. Es ist also möglich, dass jeder Netzwerkteilnehmer die gesendete Datei empfängt. Dies wird auch dadurch besonders einfach gemacht, dass immer der gleiche Port verwendet wird. Man muss daher keinen Portscan durchführen, um die Datei unberechtigt empfangen zu können. Sharik ist sowohl auf Android, i-Produkten und Windows als auch auf Linux verfügbar. Es gibt auch verschiedene Wege Sharik zu installieren. Darüber hinaus verfügt auch Sharik über die Fähigkeit mehrere Dateien auf einmal zu versenden. Dafür verwendet Snapdrop einen kleinen Trick. Er verpackt alle Dateien, welche versendet werden sollen in ein ZIP-Datei und bietet diese an.

Mein Programm kommuniziert zwar auch immer auf dem gleichen Port (ausgenommen sind Dateiübertragungen, siehe unten), allerdings ist die Kommunikation verschlüsselt und durch die Überprüfung der Prüfsumme wird sichergestellt, dass nur das richtige bzw. berechtigte Gerät auf die Datei zugreifen kann.

Mein Programm

Das Programm sollte möglichst programmunabhängig sein. Des Weiteren musste es in der Sprache geschrieben werden, welche ich beherrsche. Zusätzlich ist für eine freundliche Benutzerinteraktion auch eine graphische Schnittstelle (GUI) erforderlich. Ich habe mich für das „long-live project“ [www.ruby-toolbox.com/projects/fxruby] fxruby entschieden. Dieses basiert auf dem FOX-Toolkit, welches in C++ geschrieben wurden ist. Für die Verschlüsselung verwendet mein Programm OpenSSL, welches native Bibliotheken in Ruby hat. Der Name des Programms ist aus meinem Pseudonym „Bandura Yeo Borissowitsch“ entnommen und der Funktionalität des Programms: „Banduras Lan Messenger“. Dies werde ich im folgenden mit BLM aufkürzen.

Konzept

Eines der größten Probleme war, dass das Problem mehrere Datei auf einmal senden sollte. Das Programm führt allerdings die Hauptkommunikation über eine einzige Verbindung (Socket). Dieser Hauptkommunikationskanal wird im folgenden als Control-Socket bezeichnet.

Auftragswarteschlange

Um die „Aufträge“ zu koordinieren, habe ich ein „Warteschlange“-Prinzip implementiert.
Es gibt folgende Auftragstypen:

- Datei senden
- Nachricht senden

Datei senden. In dieser Art von Auftrag wird einmal der Dateiname und der Dateipfad gespeichert.

Nachricht senden. In dieser Art von Auftrag wird lediglich die Nachricht gespeichert.

Bei jeden Auftrag wird auch der Auftragstyp gespeichert.

Senden von Dateien

Bevor eine Datei gesendet wird, wird eine neue Verbindung über einen neuen Socket ausgehandelt. Der Client kennt bereits das Zertifikat des Servers. Nun ist es nur noch nötig, dass der Server

sicherstellt, dass er mit dem richtigen Client verbunden ist. Der Server lässt nur eine Verbindung zu. Daher kann entweder der „richtige“ oder der „falsche“ Client verbunden sein. Um dies zu überprüfen sendet der Client eine Bestätigungsnachricht über den Control-Socket. Bevor der Client die Bestätigungsnachricht sendet, überprüft dieser das Zertifikat des Servers. Dadurch wird sichergestellt, dass sowohl Server als auch Client mit dem richtigen Partner verbunden sind.

Die eigentliche Dateiübertragung erfolgt dann über die neu ausgehandelte Verbindung. Während der Aushandlung bleibt der Control-Socket für weitere Aufträge gesperrt.

Um eine maximale Datenübertragungsrate zu erreichen und das Netzwerk nur kurz, dafür aber stark, auszulasten, wird die Datei als ersten in den Arbeitsspeicher gelesen und in base64 codiert. Die base64codierte Datei wird dann übertragen. Des Weiteren wird eine Prüfsumme MD5 für die eigentliche Datei übertragen.

Der Empfänger speichert die base64codierte Datei in dem Arbeitsspeicher, dekodiert diese und speichert sie anschließend. Bevor die Datei gespeichert wird, wird der Dateiinhalt wird der übertragenen Prüfsumme verglichen. Wurde die Datei fehlerhaft übertragen, daher der Vergleich fällt negativ aus, wird die Datei trotzdem gespeichert, es wird aber eine Fehlermeldung auf im Chatverlauf ausgegeben. Der Dateiname und die Prüfsumme wird separat übertragen über die neu ausgehandelte Verbindung übertragen.

Verwirklichung meiner Idee

Anfang

Beim Start des Programms erscheint ein Initialisierungsfenster. In diesem gibt man unter anderem die Position des Gerätes an. Es gibt die Positionen Client und Server. Diese Information ist nur für den Verbindungsaufbau wichtig. Später sind beide Partner gleichberechtigt und verhalten sich bis auf die Verbindungsaushandlung gleich. Es ist daher auch egal, welches Gerät Client und welches Server ist.

Im Servermodus wird einem eine Liste von möglichen IPv4-Adressen angeboten, welche man als Server annehmen kann. Hier sollte man entsprechend jene wählen, welche das andere Gerät auch erreichen kann. Die IP-Adressen werden von der Methode `ip_address_list` aus der Klasse `Socket` entnommen (ruby-doc.org/stdlib-3.0.2/libdoc/socket/rdoc/Socket.html#method-c-ip_address_list, stackoverflow.com/questions/14112955/how-to-get-my-machines-ip-address-from-ruby-without-leveraging-from-other-ip-ad). Danach werden für das Programm unnötige IP-Adressen entfernt. Dazu können beispielsweise IPv6-Adresse oder Loopback-Adressen zählen. Interne IPv6-Adressen werden nur selten in internen Netzwerken vergeben. Loopback-Adressen sind auch für eine Kommunikation zwischen Geräten nicht zu gebrauchen.

Im Clientmodus ist es lediglich nötig, die IP-Adresse vom Server (welche zuvor in der Liste ausgewählt wurde) auszuwählen. Es ist nötig, dass der Server zuerst gestartet wird und sich danach erst der Client verbindet.

Verbindung

Die Verbindung wird mithilfe von OpenSSL geschützt. Dafür wird ein RSA-Schlüssel mit einer Stärke von 4096 Bits erzeugt (die Schlüsselstärke ist leicht anpassbar; es ist also möglich sie in Zukunft zu erhöhen). Des Weiteren wird ein X509-Zertifikat erzeugt. Diese wird mit dem RSA-

Schlüssel und den SHA512-Hashalgorihtmus signiert. Das Zertifikat ist nur zwei Tage gültig. Bei jedem Start des Programms wird ein neuer Schlüssel und ein neues Zertifikat erzeugt.

Danach wird ein SSL/TLS-Server erzeugt. Der Client wird sich mit diesem Server verbinden. Der Server ist so konfiguriert, dass er nur eine Verbindung (die vom Client) annimmt.

Integrität

Um zu verifizieren, dass die Geräte direkt miteinander Verbunden sind, wird auf beiden Geräten die SHA256-Prüfsumme des Server-Zertifikats angezeigt. Eine SHA512-Prüfsumme wäre zwar sicherer, allerdings auch länger. Dadurch, so meine Vermutung, könnte man die Prüfsumme schlechter vergleichen und man würde vielleicht ohne zu Vergleichen die Prüfsumme bestätigen. Wenn die gleiche Prüfsumme auf beiden Geräten angezeigt wird, sind sie erfolgreich miteinander verbunden. Wenn nicht, kann es sich um einen Fehler handelt. Es kann aber auch sein, dass ein Angriff (z. B. Man-in-the-Middle-Attack) durchgeführt wurde. In diesem Fall wird die Verbindung abgebrochen und man kann durch einen Neustart des Programms versuchen, die Verbindung erneut herzustellen.

Events

Als nächstes werden im Kern des Programms (`BLMSocket.rb`) verschiedene Ereignisse (Events) registriert. Es gibt die folgenden Ereignisse:

- `:receive_message`
- `:receive_file`
- `:send_file`
- `:bad_file_received`
- `:conn_reset`

Diese Ereignisse dienen dazu, dass die GUI eine entsprechende Meldung ausgeben kann. Das Programm würde auch ohne dieses System funktionieren. Das Problem wäre dann allerdings, dass der Benutzer nicht über entsprechende Ereignisse über die GUI informiert werden würde.
`:receive_message` Dieses Ereignis wird ausgelöst, wenn eine Nachricht empfangen wird. Es dient dazu, dass in der großen Textbox im Hauptfenster, die entsprechende Nachricht angezeigt wird.
`:receive_file` Dieses Ereignis wird ausgelöst, wenn eine Datei empfangen wird. Die GUI gibt dann im Chatverlauf aus, dass eine Datei empfangen wurden ist sowie den entsprechenden Dateinamen.
`:send_file` Dieses Ereignis wird ausgelöst, wenn eine Datei gesendet wird. Im Chatverlauf wird der Benutzer dann darüber informiert.
`:bad_file_received` Dieses Ereignis wird ausgelöst, wenn eine Datei fehlerhaft übertragen wurden ist. Konkret bedeutet dies, dass die übermittelte und die selbst berechnete MD5-Prüfsumme nicht gleich sind. Die Datei wird zwar trotzdem gespeichert. Der Benutzer wird aber auch darüber informiert.
`:conn_reset` Dieses Ereignis tritt auf, wenn die Verbindung zurückgesetzt wurden ist oder "das Ende der Datei" (EOF) erreicht wurden ist. Also, wenn die Verbindung mit dem Partner nicht mehr intakt ist. Der Benuztzer wird über einen Pop-Up-Dialog darüber informiert. Das Programm beendet sich daraufhin.

Hauptfenster

Nachdem die Verbindung zum anderen Gerät hergestellt wurden ist und die Integrität über die Prüfsumme bestätigt wurden ist, erscheint das Hauptfenster. Dieses ist aufgrund meine mangelten

Designer-Fähigkeiten recht stumpf, einfach und simpel gehalten. Es gibt eine große Textbox, welche nur lesbar ist. In dieser erscheinen Textnachrichten. Darunter befindet sich ein kleines Textfeld und daneben ist Button. Dieses ist zum Senden von Nachrichten da. Des Weiteren befindet sich darunter noch zwei Button. Mit dem einen kann man eine Datei senden und mit dem anderen es ist möglich, dass Download-Verzeichnis (also dort wo die empfangenden Dateien gespeichert werden) zu ändern. Standardmäßig wird das Verzeichnis, wo das Programm ausgeführt wird als Download-Verzeichnis verwendet (Pfaf: `./`).

Innenlaben und Auftragswarteschlange

Im Programm läuft im Hintergrund ein System, welches den Hauptkommunikationskanel (Control-Socket) auf empfangende Daten prüft. Wenn keine vorliegen, werden die eigenen Aufträge abgearbeitet.

Dafür gibt es eine Auftragswarteschlange. Diese kann man sich gut mit eine Analogie vorstellen: Wir sind in einem Supermarkt. Es gibt mehrere Kassen für verschiedene Produkte (die Aufträge). Für manche Produkte gibt es nur eine Kasse, für andere mehrere[1]. Des Weiteren gibt es einen „Superkassierer“, welcher die Kunden den entsprechenden Kassen zuteilt. Nach der Zuteilung, welche ggf. dauert, geht es richtig schnell an den Kassen weiter.

Es gibt folgende Typen an Aufträgen:

- Datei senden
- Nachricht senden

Datei senden. Ein Auftrag dieser Art beinhaltet zusätzlich zum Auftragstypen noch den Dateinamen und den Dateipfad.
Nachricht senden. Dieser Auftrag beinhaltet lediglich den Typ und die Nachricht. Es ist keine Angabe des Empfängers notwendig, da es nur einen in meinem Programm geben kann.

Datei senden

Wenn keine Aufträge vom Partner vorliegen, wendet sich dem Programm seinen eigenen Aufträgen zu.

Um eine Datei zu senden, wird ein weiterer Socket eröffnet. Der Port wird vom Server festgelegt und dem Client mitgeteilt. Der Client verbindet sich dann zum neuen Server. Der Server ist wieder nur so konfiguriert, dass sich nur ein Gerät darüber verbinden kann. Nach einem erfolgreichen Verbindungsaufbau prüft das Client das Serverzertifikat. Wenn es mit dem bereits bekannten Zertifikat übereinstimmt, ist der Client mit dem richtigen Gerät verbunden. Wenn es nicht übereinstimmt, bricht der er den Sendevorgang ab.

Nachdem der Client mit dem Server verbunden ist und das Zertifikat überprüft hat, sendet er ein Bestätigungsnachricht an den Server. Dies bedeutet für den Server, dass er auch mit dem richtigen Client verbunden ist. Schlägt die Zertifikatsprüfung beim Client fehl sendet er keine Bestätigungsnachricht an den Server und der Server weiß, dass er nicht mit dem richtigen Partner verbunden ist.

1 Um es etwas auszuschmücken, könnte man sagen, dass es eine Kasse für Kartoffeln gibt (diese sind nicht sehr beliebt) und dass es “unendlich” viele Kassen für Äpfel gibt (da diese beliebt sind).

Diese Bestätigungsnachricht wird über den Control-Socket gesendet. In der Zeit der Verbindungsaushandlung, Zertifikatsprüfung und Bestätigungsnachricht ist der Control-Socket gesperrt. Es kann also nicht von anderen Teilen des Programms benutzt werden.

Die eigentliche Übertragung der Daten erfolgt dann über den neuausgehandelten Kanel (Socket). Diesen bezeichne ich im folgenden als File-Socket.

Als ersten wird die Datei gelesen und in den Arbeitsspeicher kopiert. Dann wird die Datei base64codiert. Aus der Originaldatei wird außerdem eine MD5-Prüfsumme berechnet. Die base64codierte Datei, der Dateiname und die MD5-Prüfsumme werden nun über den File-Socket übertragen.

Der andere Partner empfängt die Daten und speichert diese im Arbeitsspeicher. Danach dekodiert er die Daten und berechnet auch die MD5-Prüfsumme. Wenn die übermittelte und die selber berechnete Prüfsumme übereinstimmen ist die Datei erfolgreich übertragen wurden. Die Datei wird dann gespeichert. Wenn die Prüfsummen nicht übereinstimmen, wird die Datei trotzdem gespeichert. Es wird allerdings das Ereignis `:bad_file_received` ausgelöst. Dadurch wird der Benutzer über die fehlerhaft Übertragende Datei informiert. Bereits vorhandene Dateien werden überschrieben. Dies ist beispielsweise bei Urlaubsfotos nützlich: Man übertragt die Urlaubsfotos von einem zum anderen Gerät. Nun hat man vielleicht vergessen, ob man genau dieses Foto bereits gesendet hat. Man kann es nun einfach mitsenden, da das gleiche bereitsvorhandene Foto überschrieben werden würde.

Port

Für den Control-Socket wird immer der Port 20205 verwendet. Dieser liegt auch so hoch, dass man ihn ohne zusätzliche Berechigungen benutzen darf. Des Weiteren ist es unwahrscheinlichter, dass ein anderes Programm ihn benutzt. Die File-Sockets verwenden Ports zwischen 20206 bis 20400.

Damit wäre es also theoretisch von porttechnichen Möglich $20400-20206=194$ Dateien aufeinmal zu übertragen. Das Programm speichert sich die Ports, welche für die File-Sockets verwendet werden. Wenn für eine Dateiübertragung (zufällig) ein Port ausgewählt wird, welcher bereits vergeben ist, merkt dies das Programm und wählt einen anderen. Wenn ein Port nicht mehr benutzt wird (weil zum Beispiel die Dateiübertragung abgeschlossen ist), kann dieser wieder verwendet werden.

Bibliothek

Es werden einige Bibliotheken, wie zum Beispiel OpenSSL, in BLM benutzt. Außer eine Bibliothek sind allerdings alle Bibliotheken standardmäßig in Ruby mitgeliefert. Die einzige Bibliothek, welche zusätzlich heruntergeladen werden muss, ist `fxruby`. Dabei handelt es sich um das GUI-Toolkit, welches BLM benutzt. Fxruby basiert auf dem FOX-Toolkit (fox-toolkit.org, en.wikipedia.org/wiki/Fox_toolkit), welches in C++ geschrieben wurden ist (rubygems.org/gems/fxruby).

Veröffentlichung

Ich habe ein Versionsverwaltungssystem (Git) zur Programmierung verwendet. Dies hat unter anderem den Vorteil, dass wenn man einen Fehler macht, man immer zu einer vorherigen (funktionierenden) Version des Programms zurückgehen kann. Erst habe ich BitBucket als Plattform dafür gewählt, da es für private Repository mehr Vorteile anbietet. Schlussendlich habe

ich mich allerdings dazu entschieden, dass Programm auf GitHub zu veröffentlichen. Das entsprechende Repository kann man unter https://github.com/marek22k/lanmessenger finden.

Bewertung

Nachdem ich nun mein Programm vorgestellt habe, möchte ich nun überprüfen, ob mein Programm den am Anfang genannten Kriterien entspricht.

freie Software

Ich habe mich dazu entschlossen mein Programm „gemeinfrei“ zu machen, also an allen Rechten, welche ich an meinem Programm hätte, zu verzichten. Um mehr Rechtssicherheit zu gewinnen, verwende ich die Lizenz „Do What the Fuck You Want to Public License” („WTFPL”, wtfpl.net).

Verbindung

Damit mein Programm das andere Gerät findet, wird der Benutzer dazu aufgefordert, die IP-Adresse des anderen Gerätes einzugeben. Dies sorgt für einen schnellen Verbindungsaufbau. Des Weiteren kann es keine Programme mit dem Aufspüren des Gerätes geben. Der Nachteil daraus ist allerdings eine schlechtere Benutzererfahrung.

Integrität und Verschlüsselung

Mein Programm benutzt zur Verschlüsselung OpenSSL. Um die Integrität des anderen Gerätes zu verifizieren, werden dem Benutzer auf beiden Gerätes Prüfsummen angezeigt. Wenn diese gleich sind, wurde die Integrität verifiziert.

Plattformunabhängigkeit

Es gibt verschiedene Möglichkeiten Plattformunabhängigkeit zu erreichen. Ich tue dies, indem ich seit Jahren existierende Bibliotheken sowie die Programmiersprache Ruby verwende. Jedoch kann weder Ruby ohne größere Umstände noch die GUI-Bibliothek auf Android oder iOS verwendet werden.

Mehrere Dateien senden

Die Architektur meines Programms lässt durch die „Auftragswarteschlange“ beliebig viele Aufträge zu. Daher können auch mehrere Dateien nacheinander bzw. gleichzeitig übertragen werden. Aktuell ist allerdings auch kein „Limiter“, also ein Limit für gleichzeitige Aufträge, implementiert. Dadurch könnte es vorkommen, dass Computer bzw. Netzwerke sehr stark belastet werden.

Schlusswort

Ich sehe mein Programm trotz einiger Mängel als gelungen ansehen. Dennoch ist eine ständige Weiterentwicklung und Pflege von Programmen sehr wichtig. In der Zukunft habe ich daher vor, Funktionen zu meinem Programm hinzuzufügen und Fehler zu beheben. Ich kann mir auch eine Headless-Version meines Programmes vorstellen, genauso wie eine Android-App.

Quellenverzeichnis

- FSFE (2016): “Befreie Dein Android!”, https://download.fsfe.org/advocacy/promomaterial/FYA/Fdroid-leaflet/fdroid-leaflet.de.pdf, 12.01.2022
- u/dinzan (2020): “Google is making "Nearby Sharing" Open-source. Code is now on GitHub“, https://www.reddit.com/r/chromeos/comments/ho83x8/google_is_making_nearby_sharing_opensource_code/, 12.01.2022
- Dinsan Francis (2020): „Nearby Sharing Will Come to All Chromium-based Browsers? (Edge and More)“, https://www.chromestory.com/2020/07/nearby-sharing-edge/, 12.01.2022
- https://snapdrop.net/
- https://github.com/RobinLinus/snapdrop/blob/master/LICENSE, 12.01.2022
- https://github.com/RobinLinus/snapdrop/blob/master/docs/faq.md#what-about-security-are-my-files-encrypted-while-being-sent-between-the-computers, 12.01.2022
- https://www.ruby-toolbox.com/projects/fxruby, 18.05.2021
- https://ruby-doc.org/stdlib-3.0.2/libdoc/socket/rdoc/Socket.html#method-c-ip_address_list, 12.01.2022
- https://en.wikipedia.org/wiki/AirDrop, 12.01.2022
- https://stackoverflow.com/questions/14112955/how-to-get-my-machines-ip-address-from-ruby-without-leveraging-from-other-ip-ad, 12.01.2022
- http://fox-toolkit.org/, 12.01.2022
- https://en.wikipedia.org/wiki/Fox_toolkit, 12.01.2022
- https://rubygems.org/gems/fxruby, 12.01.2022
- http://www.wtfpl.net/, 12.01.2022
- Lyle Johnson (2008): FXRuby: Create Lean and Mean GUIs with Ruby, Pragmatic Programmers (ISBN-13 978-1934356074, ISBN-10 1934356077)