INTERNATIONALE HOCHSCHULE

Prüfungsleistung: Portfolio

Projektdokumentation

Henri Wahl

Matrikelnummer: IU14077250

Studiengang: Master Informatik

Prüfungsleistung im Modul:

DLMCSPSE01_D - Projekt: Software Engineering

Sommersemester 2024

Abgabedatum: 19.7.2024

1 GitHub Repository

Das Projekt ist auf GitHub zu finden unter https://github.com/hw-iu/FileSwoosh.

2 Softwareprozess und Vorgehensweise

Da das Projekt einen sehr überschaubaren Umfang hat, bietet sich die Entwicklung nach dem Wasserfall-Modell an. Überdies ist nach Fertigstellung nicht von einer kontinuierlichen Weiterentwicklung und Wartung auszugehen, so dass z.B. agile Vorgehensweisen nicht von Vorteil sind. Ebenso wenig handelt es sich um eine Entwicklung durch ein Team, so dass Modelle, die Teamarbeit fördern, nicht präferiert werden müssen.

3 Verwendete Technologien

Programmiert wird die Applikation in **Python**, da schon Wissen darüber vorhanden ist und die Sprache eine schnelle und strukturierte Entwicklung begünstigt. Überdies ist sie sehr portabel und lässt damit die Möglichkeit offen, die Applikation auch auf anderen Plattformen als Windows zu implementieren.

Es sollen folgende Third-Party-Libraries zum Einsatz kommen:

- **PyQt6** als GUI.
- **Flask** als serverseitige Komponente für die Kommunikation der Clients, den Transport und die TLS-Verschlüsselung.
- **Httpx** als clientseitige Komponente, um auf Server-API zuzugreifen und Transport von Dateien zu initiieren. Zuerst war **Requests** vorgesehen, welches sich allerdings nicht als geeignet erwies.
- **Multicast_expert** als clientseitige Komponente, um den Host in der gemeinsamen IPv6 Multicast-Gruppe zu registrieren und über seine Existenz zu informieren
- **PyInstaller** für die Erzeugung einer unter Windows ausführbaren EXE-Datei.
- **InnoSetup** für die Erzeugung einer Setup-Datei für Windows.

Diese Technologien sollen genutzt werden:

- **QML** als GUI-Komponente von PyQt6.
- **IPv6 Multicast**, um Hosts im lokalen Netzwerk zu finden.
- **JSON** als Format für den Transport der Dateien.

4 Design- und Implementierungsentscheidungen

Im Folgenden werden Details der Implementierung erläutert:

- **Hosts finden sich per IPv6 Multicast:** Es gibt keinen zentralen Server, der alle Clients kennt und verwaltet. Stattdessen sind alle Hosts Mitglied in einer speziellen M*ulticast-Gruppe*. Jeder Host meldet sich bei allen anderen Gruppen-Mitgliedern durch Senden eines Pakets mittels des Third-Party-Moduls *multicast_expert*. Die Empfänger des Pakets verbinden sich dann per HTTPS beim Sender. Damit werden sie bei ihm registriert und stehen aus seiner Sicht zum Senden von Dateien zur Verfügung. Da jeder Host so vorgeht, benachrichtigen und registrieren sich alle gegenseitig.
- **Jeder Host ist Client und Server:** Da jeder Host Dateien senden und empfangen kann, hat er die Rollen Client und Server inne. Dies ist auch der Tatsache geschuldet, dass es sich um eine dezentrale Architektur handelt. Für die Client-Rolle kommt das Third-Party-Modul *httpx* zum Einsatz. Ursprünglich war die Verwendung von *requests* geplant. Es stellte sich jedoch heraus, dass *requests* nicht mit IPv6-URLs umgehen kann, die die Scope-ID einer Link-Local-Adresse enthalten. Für die Server-Rolle wird das Third-Party-Modul *flask* verwendet.
- **Die Dateiübertragung findet per HTTPS statt:** Die Verwendung eines HTTP-Clients und HTTP-Servers ist der angestrebten Verwendung von *HTTP* bzw. dessen verschlüsselter Variante *HTTPS* geschuldet. *HTTP* ist seit Jahrzehnten als Transport-Protokoll im Internet bewährt und funktioniert plattformübergreifend. *HTTPS* ermöglicht es dazu, die Verbindung zwischen den Hosts zu verschlüsseln. Dabei werden bei jedem Start der Applikation neue Schlüssel und Zertifikate generiert, um ihrer dezentralen Natur gerecht zu werden und ohne Zertifizierungsstelle auszukommen. Die Client-Komponente verzichtet daher auf eine Zertifikatsprüfung.
- **Die grafische Oberfläche wird mit Qt6/QML realisiert:** *Qt* und damit seine neueste Version *Qt6* ist ein seit Jahrzehnten gereiftes GUI-Framework, welches plattformübergreifend verwendet werden kann und immer wieder neuen Erfordernissen angepasst wurde. Eine dieser Anpassungen ist die Möglichkeit der Beschreibung von Applikationen in *QML*, der *Qt Modeling Language*. Um die Applikation etwas zeitgemäßer zu gestalten, wurde diese anstatt der klassischen Widgets ausgewählt. Konkret wird Qt6 hier in Form des Third-Party-Modules *PyQt6* verwendet.
- **Verwendung von Qt Threads, Signals und Slots:** Da ein Host mehrere Rollen auszufüllen hat, die unabhängig voneinander sind, werden diese innerhalb von *Qt Threads* ausgeführt. So gibt es den *DiscoveryThread*, der permanent die IPv6 Multicast-Gruppe über die Existenz des Hosts informiert und die erkannten Hosts verwaltet. Im *ServerThread* läuft ein *flask*-Server, der auf Anfragen von Clients wartet. Der *ClientThread* realisiert die Anfragen an die entfernten Hosts per *httpx*. Die *GUI* wird von der Instanz *gui* der Klasse *GUI* im main-Thread realisiert. Alle Threads kommunizieren mit *Qt Signals* und *Qt Slots* untereinander. So benachrichtigt zum Beispiel der *DiscoveryThread* die *GUI*, wenn es

Änderungen bei den bekannen Hosts gibt. Dabei wird das *Qt Signal* '*signal_hosts_updated'* gesendet. Mit diesem ist der *Qt Slot 'slot_hosts_updated'* verbunden, welcher ein Update in der GUI durchführt.

Verwendung von IPv6: Nunmehr 26 Jahre nach Veröffentlichung von IPv6 als offizieller Nachfolger von IPv4 sollte es kein Problem darstellen, es auch praktisch zu verwenden. Somit wird hier von der Verwendbarkeit einer IPv6 Multicast-Gruppe ausgegangen. Betriebssystemseitig ist nicht mit Problemen zu rechnen, da alle weit und weniger verbreiteten Systeme darauf vorbereitet sind. Ebensowenig sollten Heimnetze damit Probleme haben, da auch hier in den letzten Jahren einiger Fortschritt zu verzeichnen ist. Eher ist anzunehmen, dass größere lokale Netzwerke nicht vorbereitet sind, was hier in Kauf genommen wird. Es sind 2 Fallback-Mechanismen implementiert: ausschließliche Verwendung von Link-Local-Adressen aus dem Bereich *fe80::/64* und die manuelle Eingabe der Adressen von Hosts.

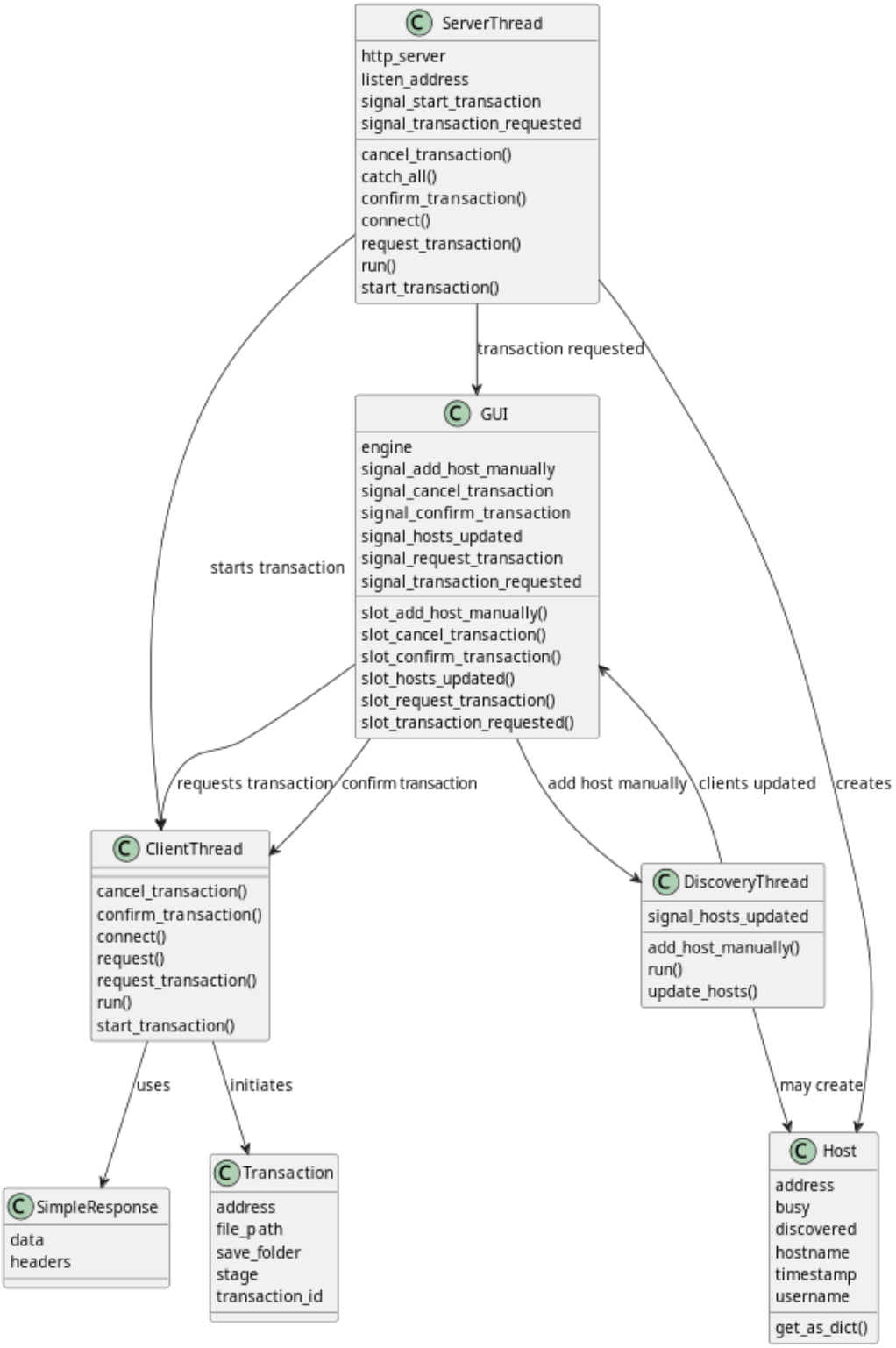
- **Verwendung von UDP und TCP an Port 56934:** IPv6 Multicast findet per UDP statt, der HTTP-Server lauscht an einem TCP-Port. Um es einfacher zu gestalten, wird für beide Zwecke der gleiche Port aus dem Bereich der dynamischen und privaten Ports verwendet, der nicht bei der IANA registriert werden muss. Es handelt sich um den willkürlich ausgewählten Port 56934.
- **Multicast-Gruppen-Adresse ff05::dead:beef:cafe:de66:** Als Adresse für die Multicast-Gruppe, in der sich alle für Dateiaustausch bereiten Hosts registrieren, ist die Adresse *ff05::dead:beef:cafe:de66* ausgewählt worden. Der Multicast-Scope *ff05::/64* wurde ausgewählt, da er das komplette lokale Netz mit Subnetzen umfasst. Die Multicast-Gruppen-ID *dead:beef:cafe:de66* ist zur Vermeidung von Kollisionen willkürlich gewählt. Dabei ist das letzte Segment die Portnummer in Hexadezimalschreibweise.

5 Design Pattern

Das hier gemäß Aufgabenstellung vorgestellte Design Pattern ist das Singleton. Die an verschiedenen Stellen im Code benötigten Konfigurationseinstellungen befinden sich im Modul *config.py.* Dieses wird in jedem Modul nach Bedarf importiert. Bei einem solchen mehrfachen Import sorgt Python per se dafür, dass es sich um dieselbe Instanz handelt, die beim ersten Import erzeugt wurde. Somit ist dafür gesorgt, dass überall die identischen Informationen vorliegen.

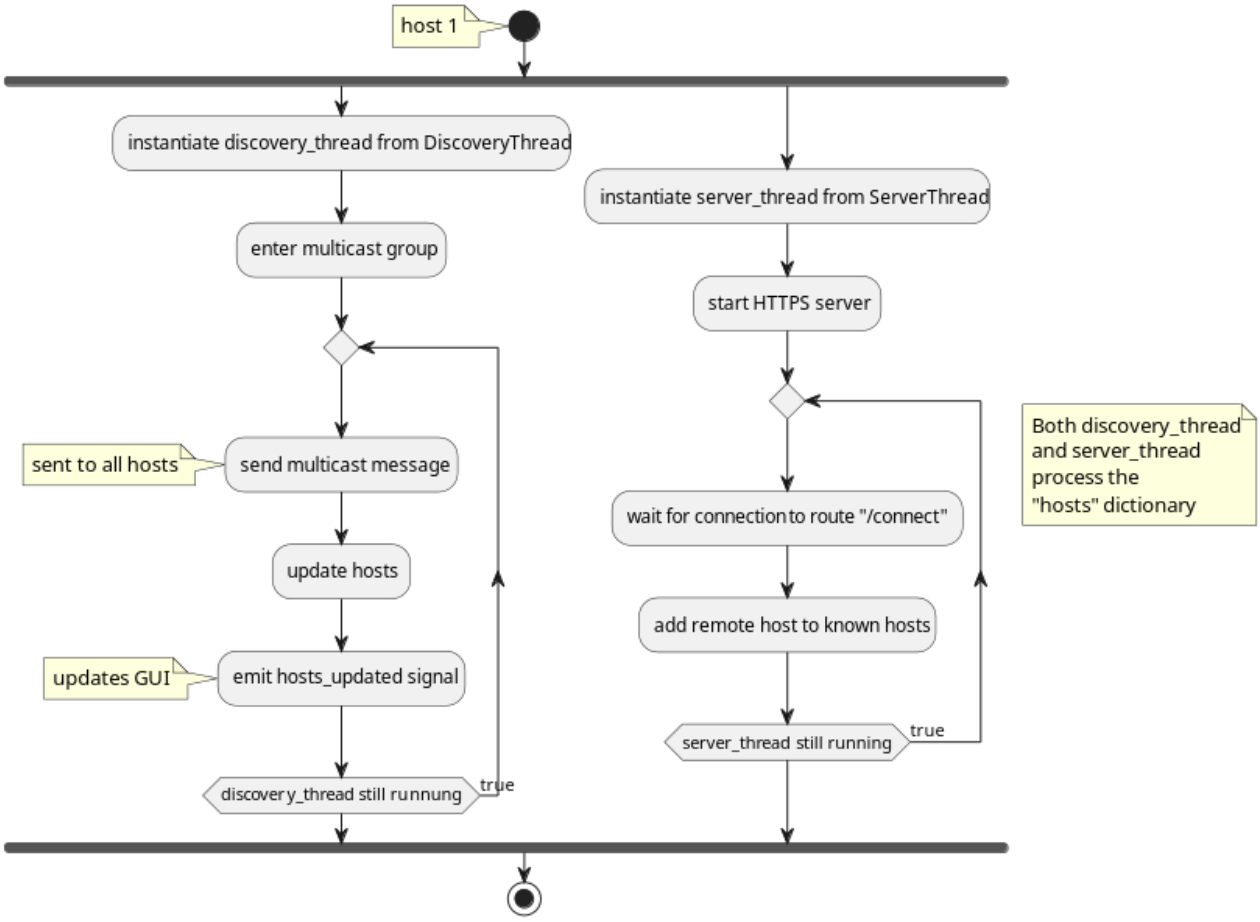
6 Struktur

Es ist wichtig zu berücksichtigen, dass die Applikation auf jedem Host gleichzeitig die Funktion von *Server* und *Client* innehat. Der dezentrale Ansatz ist ihr zentrales Element und für den Dateiaustausch wird auf externe Infrastruktur verzichtet. Das bedeutet, dass neben den Funktionen Server und Client eine dritte unverzichtbar ist: die permanente Suche potenzieller Kommunikationspartner. Dabei handelt es sich um andere Hosts, auf denen die Applikation ebenfalls läuft. Diese Suche wird hier als *Discovery* bezeichnet.

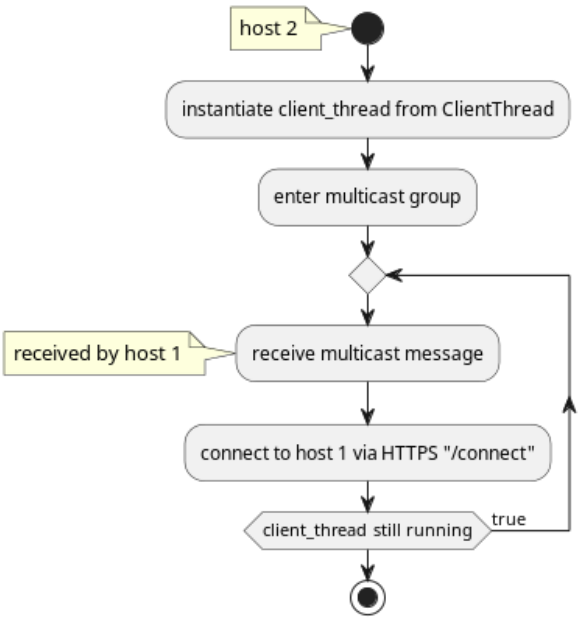


7 Weitere UML-Diagramme

Ein Kernprozess ist die Erkennung von Hosts im Netzwerk, um potenziell mit ihnen Dateien austauschen können. In den Prozess der Erkennung sind alle Backend-Threads mit einbezogen und alle laufen so auf allen beteiligten Hosts. Exemplarisch sei hier die Erkennung dargestellt, wie sie auf einem Host 1 abläuft, der sie per Multicast-Messages im DiscoveryThread initiert. Die Antwort wird ein entfernter Host 2 durch dessen *ClientThread* über die Route '/*connect*' im hiesigen *ServerThread* liefern und sich somit in der Liste der hier bekannten Hosts registrieren:



Auf Host 2 wird die Multicast-Message empfangen und zum Anlass genommen, sich bei Host 1 per HTTPS zu registrieren:



8 Benutzeranleitung

Im Folgenden werden Installation und Benutzung der Applikation vorgestellt.

8.1 Installation

Um die Windows-Firewall korrekt für die Nutzung zu konfigurieren, wird die Verwendung des Windows Installers in Form der Datei *FileSwoosh_Setup.exe* empfohlen. Sollte es nicht möglich sein, den Installer mit Administratoren-Rechten auszuführen, ist auch die Verwendung der Standalone-Variante *FileSwoosh.exe* möglich. Dieser muss möglicherweise manuell die Öffnung der Windows-Firewall gestattet werden.

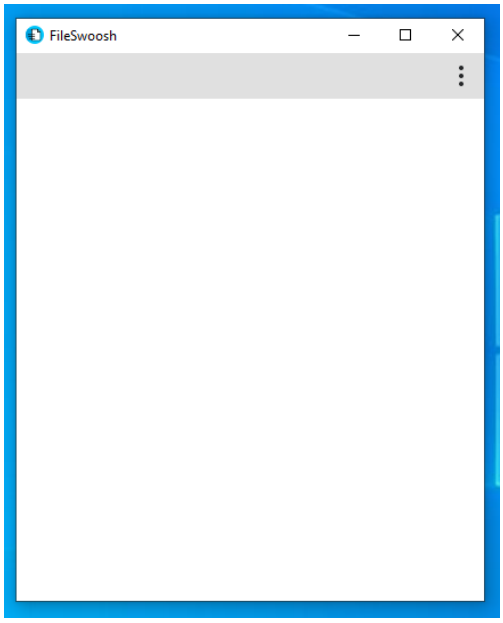
Zunächst ist der Download von https://github.com/hw-iu/FileSwoosh/releases durchzuführen. Die Ausführung des Installers *FileSwoosh_Setup.exe* wird ziemlich wahrscheinlich trotz digitaler Signatur zu einer Warnung führen:

Diese kann durch Klick auf "Weitere Informationen" und anschließendem "Trotzdem ausführen" zur Kenntnis genommen werden:

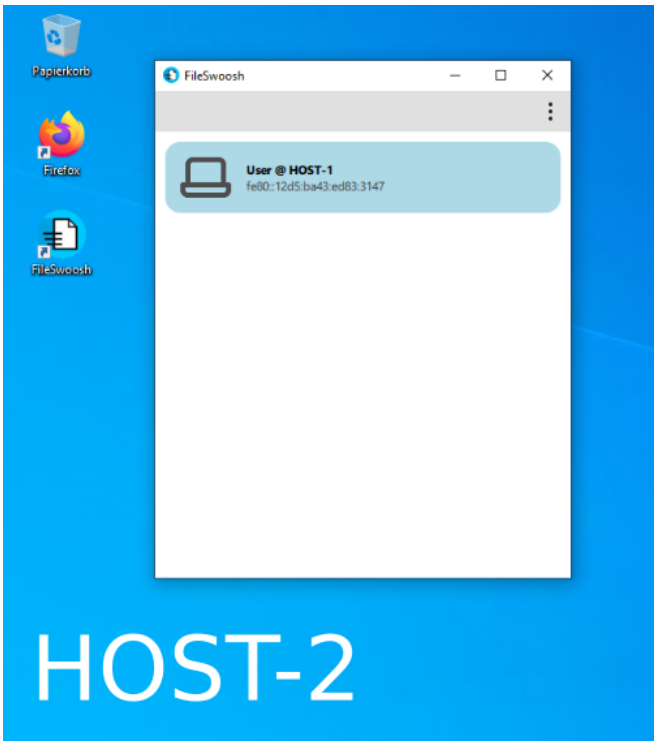
Nach erfolgter anschließender Installation per unter Windows üblichem Verfahren erscheint das Icon der Anwendung auf dem Desktop und sie steht zu Verwendung bereit:

8.2 Benutzung

Da die Anwendung dem Austausch von Dateien zwischen Hosts dient, muss sie auf mindestens 2 Hosts laufen. Für die Erläuterungen in dieser Anleitung werden beispielhaft die beiden Hosts HOST-1 und HOST-2 genutzt. Läuft sie nur auf 1 Host, so ist die Liste der gefundenen Hosts leer, da sie sich nicht selbst zum Austausch zur Verfügung steht:

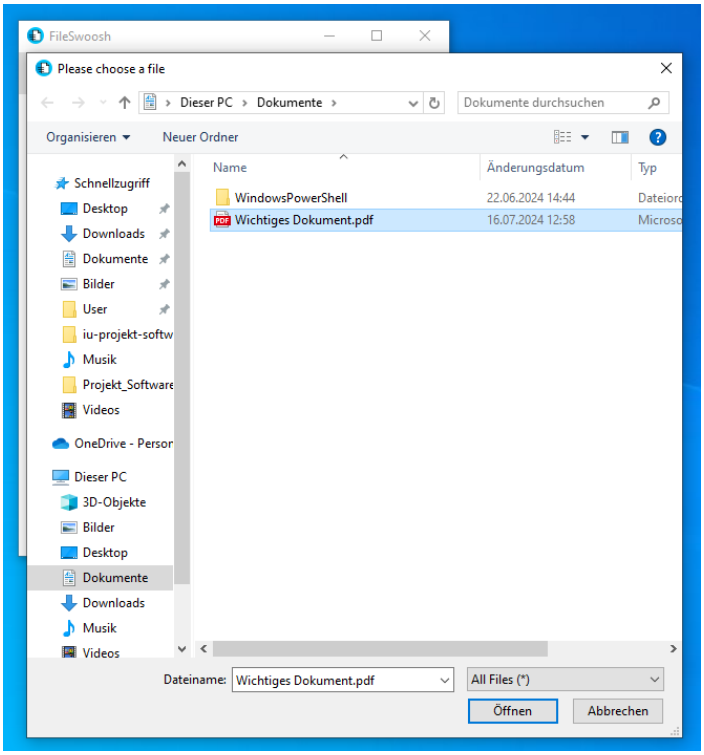


Sobald sie auf einem 2. Host gestartet wird und die Erkennung im lokalen Netzwerk funktioniert, zeigen sich die Hosts gegenseitig an:

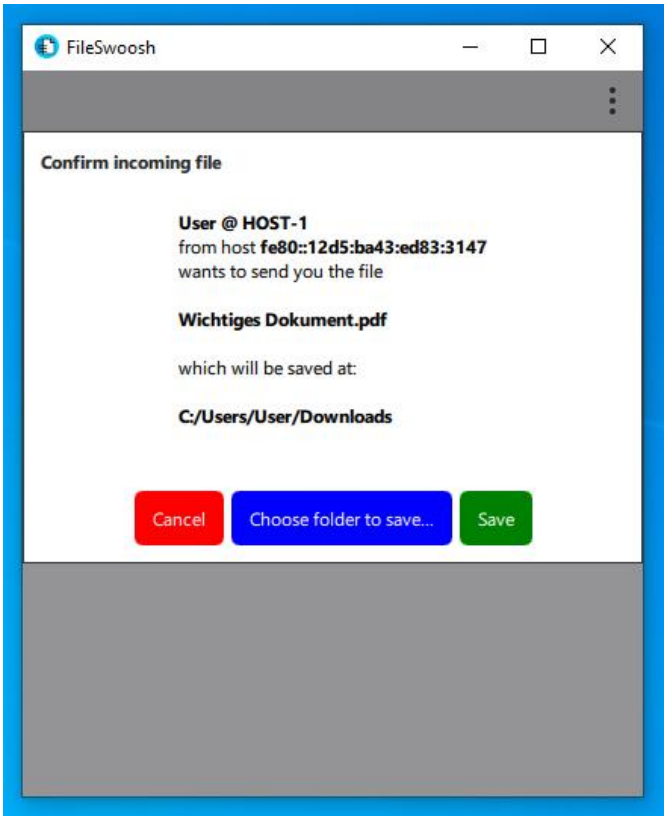


Auf HOST-1 wird HOST-2 erkannt und angezeigt und auf HOST-2 wird HOST-1 erkannt und angezeigt.

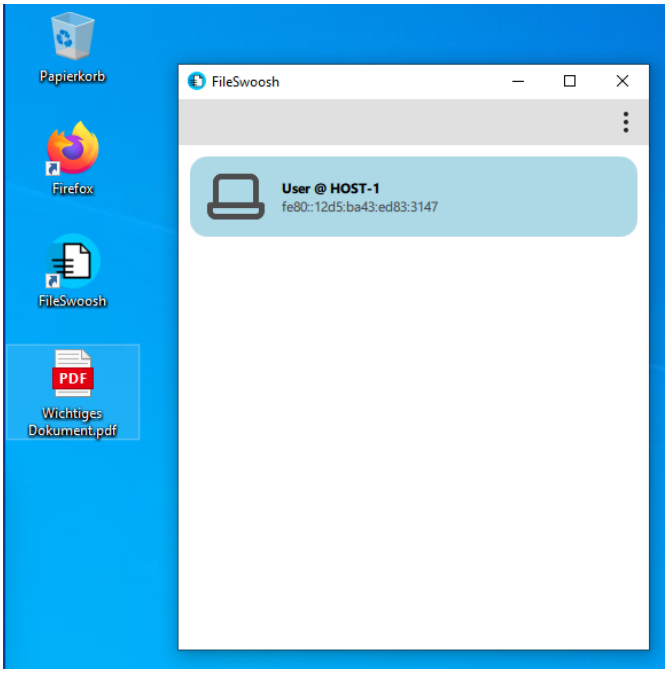
Soll nun von HOST-1 eine Datei an HOST-2 übertragen werden, so muss HOST-2 in der Liste der erkannten Hosts auf HOST-1 angeklickt werden und es öffnet sich der unter Windows übliche Dialog zur Auswahl der zu sendenden Datei:



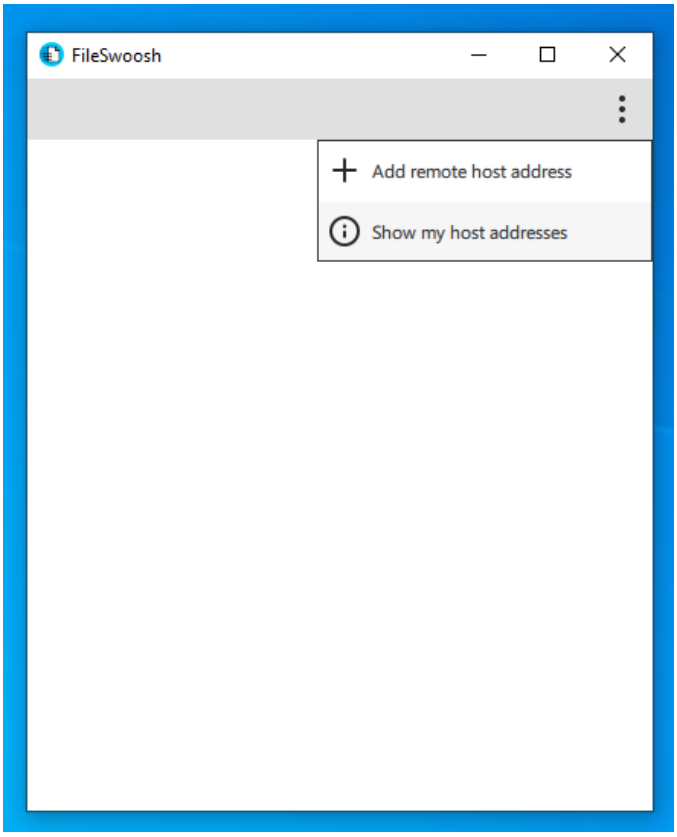
Nach Bestätigung nimmt HOST-1 mit HOST-2 Kontakt auf und fordert eine Übertragung an. Auf HOST-2 erscheint ein Dialog, mit dem die Übertragung verweigert oder akzeptiert werden kann. Außerdem kann noch ein anderer Speicherort ausgewählt werden als der standardmäßige Download-Ordner:



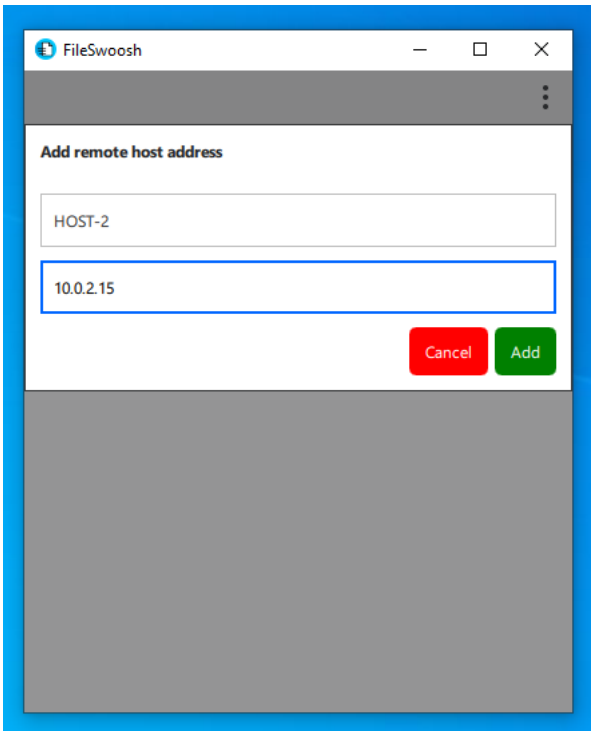
Wird ein anderer Ordner ausgewählt, wird er im Dialog aktualisiert und nach Bestätigung der Übertragung dort abgespeichert:

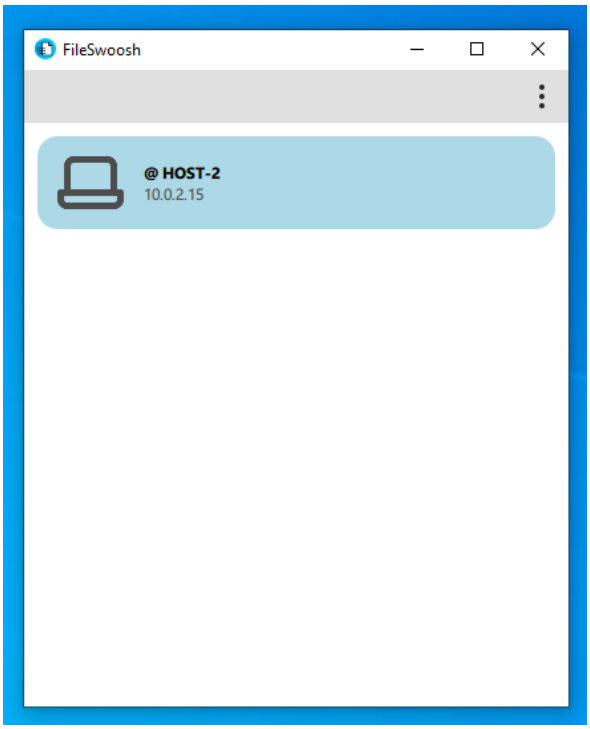


Ist es nötig, die Hosts gegenseitig manuell miteinander bekannt zu machen, kann das geschehen über das Menu:



Die Adressen der Hosts lassen sich je lokal anzeigen, um sie dem Kommunikationspartner mitzuteilen. Dieser kann sie dann bei sich hinzufügen:





Damit beide Hosts miteinander kommunizieren können, muss das manuelle Hinzufügen bei beiden durchgeführt werden.