INTERNATIONALE HOCHSCHULE

Prüfungsleistung: Portfolio

Abstract

Henri Wahl

Matrikelnummer: IU14077250

Studiengang: Master Informatik

Prüfungsleistung im Modul:

DLMCSPSE01_D - Projekt: Software Engineering

Sommersemester 2024

Abgabedatum: 19.7.2024

1 Making of

Die Idee, die Hosts sich gegenseitig finden zu lassen, führte schnell zum Einsatz von Multicast. Dank dieser Technologie können alle potenziellen Austauschpartner direkt benachrichtigt werden. Auch der Einsatz von IPv6 war mir von Anfang an wichtig, weil dieses Protokoll allmählich im Alltag angekommen ist. Zudem ist dort der Adressraum weitaus größer als bei IPv4, sodass mit weniger Kollisionen bei der Verwendung der Multicast-Gruppen-ID zu rechnen ist.

Für die Datenübertragung gab es eine Anforderung und mehrere Optionen. Die Anforderung war die, dass die Übertragung verschlüsselt ablaufen sollte, was heute eine Selbstverständlichkeit ist. Als Optionen standen die Verwendung eines eigenen Protokolls auf Sockets-Basis, von SSH und von HTTPS im Raum. Das eigene Protokoll hätte den größten Aufwand und die geringste Sicherheit gebracht, da es nicht so gut getestet wäre wie die teils jahrzehntelang erprobten Alternativen. SSH wäre da schon ein Stück sinnvoller gewesen, allerdings macht seine Architektur mit Authentifizierung vom Client gegenüber dem Server das Ganze unnötig komplex. Am sinnvollsten erschien die Verwendung von HTTPS, da es ohne Authentifizierung auskommt, eine verschlüsselte Kommunikation ermöglicht und für den Transport von Dateien optimal ist.

Die Discovery wurde durch die Kombination von Multicast-Nachrichten und darauffolgender Registrierung der Hosts beim Sender per HTTPS auch sicherer, weil so ein Host erst allgemein unverschlüsselt bekannt gibt, dass er zu den potenziellen Austauschpartnern gehört, und erst dann verschlüsselt per HTTPS von den anderen Hosts Informationen wie Nutzernamen und Hostnamen mitgeteilt bekommt.

Der finale Austausch schließlich wurde zunächst durch die Kombination von *requests* auf der Clientseite und *flask* auf der Serverseite realisiert. Beide Module sind sehr etabliert in der Python-Welt. Die Kommunikation der beiden ließ sich schnell implementieren. Leider stellte sich bei den Arbeiten für das Szenario, wenn ein Netzwerk IPv6 nicht voll unterstützt, heraus, dass *requests* nicht mit URLs umgehen kann, die die Scope-ID in einer Link-Local-Adresse enthalten. Ein Beispiel dafür ist *https://[fe80::1234:56ff:fe79:1011%3]*, wo der URL-Teil nach dem *%* zu Fehlern führt. Die Rettung war *httpx*, welches dieses Problem nicht hat. Ein positiver Nebeneffekt der Verwendung von HTTPS ist, dass die Übertragung nur komplett und fehlerfrei abgeschlossen wird und somit die angestrebte Überprüfung auf korrekte Übertragung ohne weiteres Zutun inklusive ist.

HTTPS wurde auch als sicher genug eingeschätzt, trotzdessen dass die Serverzertifikate bei jedem Start der Applikation neu erzeugt werden und keine Zertifizierungsstelle involviert ist. Da die Verbindungen eher kurzlebig sind und Austauschpartner im Zweifelsfall über die IP-Adressen gegenseitig identifizieren können, besteht wenig Gefahr, dass ein Angreifer sich als man-in-the-middle in die Kommunikation einklinkt.

Beim Thema GUI hat mich vor allem die Verwendung von QML interessiert, da ich Qt Widgets schon kannte und QML die modernere Variante ist. Überraschend war, dass in QML Einiges an Logik in

der GUI-Definition stattfindet. Zugegebenermaßen hat die Anwendung nicht allzuviel GUI, aber selbst das Wenige hat schon zu Erkenntnissen geführt.

2 Lessons learned

Die erste Lektion ist, dass der Zeitplan falsch gewichtet war. Es spielte etwas Wunschdenken eine Rolle bzw. eine Unterschätzung der auftretenden Probleme. Letztere traten nur teilweise in den verwendeten Third-Party-Modulen wie *requests* auf, denn mit der Software selbst hatte ein großes und zeitfressendes Problem nicht direkt zu tun: die Windows Firewall bzw. deren Konfiguration. Es hat eine gute Weile gedauert, bis es zu einer zufriedenstellenden Lösung gekommen ist.

Aus Letzterem ergibt sich direkt die zweite Lektion. Sie lautet, dass bei Software für Windows mit etwas mehr administrativem Aufwand zu rechnen ist. Das Deployment selbst ist z.B. mit InnoSetup noch vergleichsweise einfach realisierbar gewesen und im Grunde einfacher als eine korrekte Paketierung in Linux per Debian, RPM oder vor allem Flatpak. Jedoch das Ermitteln der notwendigen Freigaben in der Firewall und das Einflechten in den Installer waren mühsam. Möglicherweise wäre das bei häufigerer Entwicklung von Software dieser Art für Windows durch mehr Übung ein geringeres Problem.

Eine dritte Lektion ist, dass es trotz gleicher Technologien einen Unterschied macht, ob die Kommunikation nur auf einem physischen Host zwischen mehreren VMs stattfindet oder ob sie tatsächlich durch ein Netzwerk zwischen physischen Hosts stattfindet. Die in der Virtualisierung so reibungslosen Abläufe kamen in der Realität leider etwas ins Stocken.

Eine vierte Lektion ist, dass bei der Dateiübertragung die Möglichkeiten der Nebenläufigkeit von Code, wie sie auch *httpx* bietet, in Betracht gezogen werden sollten. Trotz der Verwendung von Threads kann es in der GUI bei der Übertragung großer Dateien zu Rucklern kommen, die ich so nicht erwartet habe.

3 Fazit

Insgesamt empfand ich es als eine spannende Herausforderung, in kurzer Zeit eine lauffähige Software und ihre Dokumentation herzustellen. Es war eine gute Gelegenheit, einige schon vertraute Technologien neu zu kombinieren und teilweise Alternativen kennenzulernen, wie etwas das neuere *httpx* anstatt *requests*. Zudem war es ein guter Anlass, *QML* einmal näher zu beleuchten. Selbst die mühselige Beschäftigung mit dem Windows-Deployment hat zu Erkenntnissen geführt, die sicher einmal verwendbar sein werden, und sei es nur eine großzügigere Zeitplanung.