SDIM = **S**icheres **D**ezentrales **I**nstant **M**essaging

Aaron Nees, Fabian Ruf, Manuel Laug, Thomas Boch

1. Zusammenfassung

Viele proprietäre Applikationen zum Austausch von Nachrichten basieren auf dem Protokoll XMPP. Allerdings werden die freie Auswahl des Chat-Clients und die Kompatibilität mit anderen Chat-Diensten meist absichtlich eingeschränkt. Unsere Projektidee ist es, mittels XMPP und Tor, einen sicheren und dezentralen Nachrichtenaustausch über einen Raspberry Pi zu ermöglichen, während die freie Auswahl eines Account-Anbieters (z. B. mail.de) und des Clients (z. B. Conversations) möglich ist.
Um den Umgang mit diesen Technologien zu erleichtern, wollen wir einen webbasierten Einrichtungs-Assistenten erstellen, der das Einrichten und Hosten eines XMPP-Servers vereinfacht und für jeden Besitzer eines Raspberry Pi zugänglich macht.

Weitere Ideen sind das Ansprechen verschiedener Hardware-Schnittstellen (z. B. um Auslastung, Server-Aktivität, oder Nutzerzahlen anzuzeigen) sowie das Implementieren eines Chat-Bots. Über diesen kann mit anderen Geräten im Heimnetz des Raspberry Pi interagiert werden (z. B. der FRITZ!Box und ihrer Schnittstelle für Home-Automation) und es können Daten aus den Hardware-Schnittstellen ausgelesen werden.

2. Grundlegende Technologien

Das Protokoll XMPP (ehemals Jabber), über welches das Instant Messaging ermöglicht werden soll und das Tor-Netzwerk, in dessen Umfeld sich der XMPP-Server befinden soll, sind die Hauptbestandteile der Projektidee.
Die Architektur von XMPP ist vergleichbar mit der des E-Mail Dienstes. Jeder hat die Möglichkeit einen eigenen XMPP-Server zu hosten und Nachrichten nach Belieben in einem internen Netz, oder mit öffentlich erreichbaren Servern austauschen. Die Nachrichten können dabei über verschiedene Verschlüsselungsverfahren Ende-zu-Ende verschlüsselt werden. Zusätzlich werden die Metadaten, die neben dem Inhalt einer Nachricht anfallen, durch die Nutzung des Tor-Netzwerkes verschleiert.

2.1 XMPP (= E**x**tensible **M**essaging and **P**resence **P**rotocol)

Zur Realisierung des Projekts ist ein XMPP-Server erforderlich, welcher auch ausgehend von der geplanten Webschnittstelle ansprechbar sein muss. Als XMPP-Server kommen dafür mehrere Produkte in Frage (hierbei werden nur einigermaßen bekannte und frei verfügbare Server betrachtet):

ejabberd (GPL-Lizenz)	https://www.ejabberd.im/
Openfire (Apache Lizenz 2.0)	https://www.igniterealtime.org/projects/openfire/
Prosody IM (MIT-Lizenz)	https://prosody.im/
Tigase (GPL-Lizenz)	http://www.tigase.net/

Präferenz: Prosody IM

Vorteile der Nutzung von Prosody IM:

- Erfahrung mit dem Server im Team vorhanden
- Nicht zu kompliziert in der Einrichtung (Die Einrichtung von ejabberd ist relativ kompliziert)
- Erweiterbar durch Module
- Hohe Abdeckung von XEPs (XMPP Extension Protocols)
- Performanter und leichtgewichtiger XMPP-Server https://gist.github.com/hassy/9731688 (im Test wird eine "alte" Prosody-Version verwendet)
- Verfügbar für die ARM-Architektur
- Aktive Community
- Relativ sichere Standardeinstellungen (TLS erzwungen, kein unverschlüsselter Verbindungsaufbau (TLS))

Nachteile der Nutzung von Prosody IM:

- Installation von "Community-Modulen" nötig -> Module haben teilweise Alpha-Status
- Nicht clusterfähig

2.2 Tor (= **T**he **O**nion **R**outer)

Die Tor-Integration erfordert die Reservierung bzw. Generierung einer Tor-Domain. Dies geht mit dem Besitz eines privaten Schlüssels einher, welcher den Anspruch an der Tor-Domain regelt: https://www.torproject.org/docs/hidden-services.html.en

Die dadurch generierte ".onion"-Domain kann dann als XMPP-Domain eingesetzt werden.

Die daraus resultierende JID (Jabber-ID) könnte beispielsweise folgendermaßen aussehen *"max@f45ls8sqhsw7gwo4.onion"*

Herausforderungen:

- TLS über Tor erfordert die Nutzung selbstsignierter Zertifikate -> nur wenige CAs stellen valide Zertifikate für die .onion-TLD aus. Teilweise werden bei Ausstellung die Kriterien/Voraussetzungen für Extended Validation gefordert (für das Projekt kaum zu erfüllen).
- Verbindungen zu .onion-Domains nur aus dem Tor-Netzwerk bzw. anderen SDIM-Instanzen möglich
- Auch Sender-Server muss Tor-Konnektivität haben und .onion "routen" können
- Tor Hidden Services müssen sorgsam konfiguriert werden (https://blog.torproject.org/blog/hidden-services-need-some-love)
- Da in Verbindung mit Tor kein klassisches DNS zur Verfügung steht, kann der Einsatz von SRV-Records für XMPP nicht stattfinden. Hier muss auf die Standardports für XMPP gesetzt werden.
- Die SDIM-Instanz sollte keine "Leaks" in das "übliche" Internet verursachen, welche eine Enttarnung des Diensts verursachen könnten. Der Abruf aus dem Internet sollte sich auf Updates des Systems beschränken, welche ggf. ebenfalls über Tor realisiert werden können (hier werden allerdings Exit-Nodes passiert).
- Auch Informationen über den Server (Version, Laufzeit, etc.) sollten weitestgehend nicht von außen abrufbar sein, da dadurch auf den verborgenen Server geschlossen werden kann.
- .onion-Domains sind schwer zu merken (Mechanismen wie QR-Codes können zur sicheren Vereinfachung genutzt werden)

- Nützliche XEPs (XMPP-Extensions) sollten in der Standardkonfiguration aktiviert sein (auch eine mobile Nutzung des Servers durch Clients soll ermöglicht werden)

Vorteile der reinen Nutzung von Tor:

- Umgehung möglicherweise kompromittierter Tor-Exit-Nodes (bezüglich XMPP)
- Nur Verbindungen im Tor-Netzwerk möglich (können auch ohne Zertifikate als sicher angesehen werden)
- Zusätzliche Zertifikate erhöhen die Sicherheit (server2server- und client2server-Verbindungen zusätzlich transportverschlüsselt)
  -> Selbstsignierte Zertifikate können allerdings Risiken bergen und mindern die Usability des Nutzers (Fehlermeldungen in Clients)
- Probleme mit DNS und insbesondere mit NAT an dynamischen Privatnutzer-WAN-Verbindungen werden umgangen

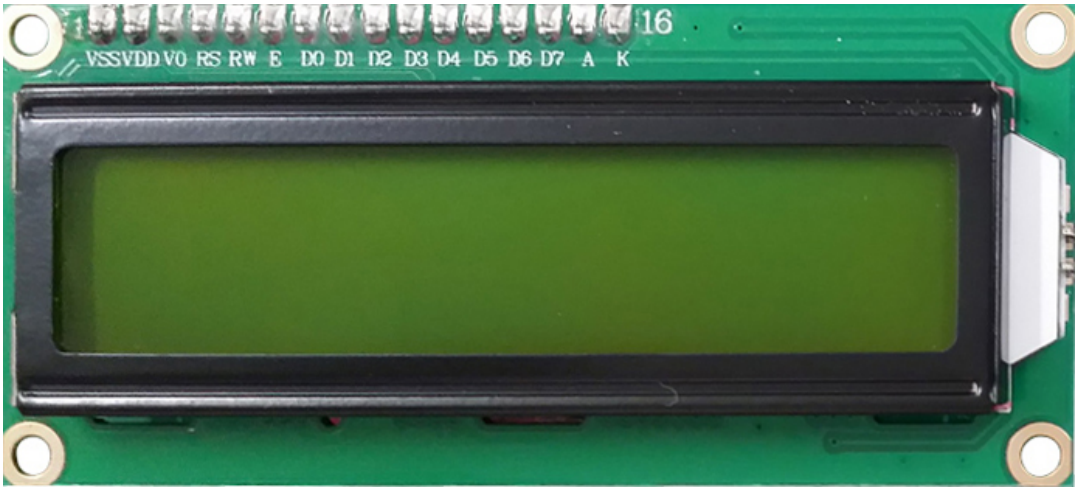
3. Hardware

Die Funktionalität des SDIM-Projekts ist grundsätzlich durch die Software gelöst und hat somit keine besonderen Anforderungen an die verwendete Hardware.
Im Rahmen des IoT-Projekts wird das Raspberry Pi in der Version 3 als Grundlage gewählt und somit die komplette Anwendung auf den Betrieb darauf ausgelegt.
Dabei werden aber keine speziellen Funktionen des Raspberry Pi 3 verwendet (wie z. B. Wifi oder Bluetooth), weshalb auch die Verwendung eines Raspberry Pi's der vorherigen Versionen 1 oder 2 keine Schwierigkeiten bereiten soll.

Die Verwendung eines Raspberry Pi's hat dabei einige Vorteile gegenüber anderer Hardware:

- Die Hardware ist weit verbreitet und Open Source
- Die Kosten sind gering
- Durch die geringe Stromaufnahme kaum Kosten im Betrieb
- Vorhandene Schnittstellen (z. B. GPIO, I2C, SPI, ...) können genutzt werden

Damit dem Anwender beim Betrieb des SDIM-Servers die Möglichkeit gegeben wird, ohne zusätzlichen Bildschirm, Informationen über das System zu sehen, wird über die GPIO-Schnittstelle ein 16x2 LCD-Display angeschlossen.



Quelle: https://www.makerlab-electronics.com/my_uploads/2016/06/16x2-lcd-i2c-1.jpg

Dieses LCD soll dazu folgende Informationen anzeigen können:

- lokale IP-Adresse, über welche das Webinterface erreichbar ist
- Onion-Adresse aus dem Tor-Netzwerk, über die der Zugriff auf den XMPP-Server ermöglicht wird
- Weitere Informationen, die über das Webinterface konfigurierbar sein sollen

Die Ansteuerung des Displays erfolgt dann über die Verwendung von Bibliotheken der entsprechenden Programmiersprache. An dieser Stelle muss darauf hingewiesen werden, dass das Display kein zwingender Bestandteil ist, um das Projekt zu betreiben. Jedoch erleichtert es den Umgang mit der Umgebung erheblich, da die lokale Adresse sonst auf einem anderen Weg herausgefunden werden muss (z. B. Webinterface des Routers).

4. Grundlagen Linux-Image

Da als Hardwareplattform das Raspberry Pi 3 gewählt wurde, ist es notwendig, dass ein Image erstellt wird, welches alle benötigten Funktionen beinhaltet. Es gibt unterschiedliche Möglichkeiten, ein solches Image zu erstellen. Zwei grundlegende Vorgehensweisen sind hier zu nennen: buildroot und debootstrap.

Buildroot bietet dabei vor allem Vorteile im Bereich Konfigurierbarkeit und Minimalismus. Dazu wird das zu bauende Image über Konfigurationsdateien beschrieben und anschließend komplett aus den Paketquellen kompiliert. Dies führt zu einem besonders stark angepassten Image, welches nur die wirklich benötigten Pakete beinhaltet. Da das komplette Betriebssystem dabei aber aus den Paketquellen kompiliert wird, entsteht am Ende ein nicht immer direkt kompatibles System gegenüber anderen Raspberry Pi Distributionen. Deshalb sind Pakete, die für eine andere Distribution erstellt wurden, nicht immer direkt kompatibel und es entsteht ein zusätzlicher Aufwand, die gewünschten Anwendungen zu installieren und zu betreiben.

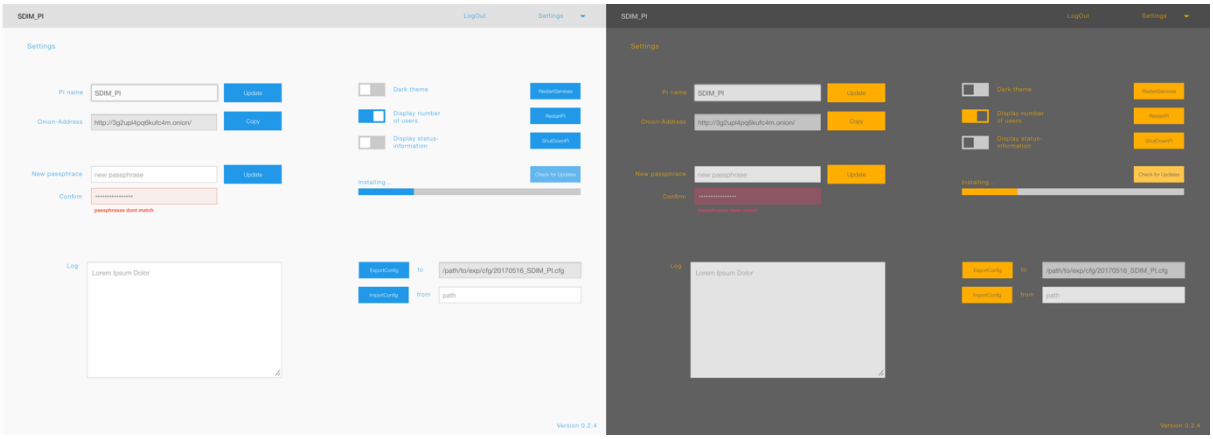
Der zweite Weg, die Verwendung von debootstrap, setzt dagegen auf ein auf Debian basierendes System auf und erstellt mithilfe von QEMU (Emulator, um ein Image für die ARM Architektur unter einem X86 System zu erstellen) ein Image auf Basis der angegebenen Paketquellen. Für dieses Projekt bietet es sich dabei an, die Paketquellen von Raspbian zu verwenden. Somit werden die gewünschten Pakete nicht kompiliert, sondern nur aus dem Repository heruntergeladen und in das Image integriert. Als Ergebnis entsteht ein zu Raspbian kompatibles Debian-Image, welches jedoch etwas größer ausfällt, als bei Buildroot. Der Grund sind die zusätzlichen Systembestandteile wie ein vollwertiges Init-System, der Paketmanager und sonstige Abhängigkeiten. Die Modifikationen an diesem Image, um die Anwendung betreiben zu können, werden über einfache Shell-Skripte vorgenommen und sind somit vollständig transparent.
Aufgrund der größeren Flexibilität soll im Rahmen dieses Projekts deshalb das Image auf Basis von Debootstrap erstellt werden. Dazu gilt es im ersten Schritt ein Skript zu erstellen, welches das grundlegende Image erstellen kann. Das Image muss alle grundlegenden Systemkomponenten, wie Kernel, Firmwaredateien, Paketmanager, Init-System, usw. beinhalten. Das Skript wird anschließend mit weiteren Befehlen erweitert, damit die Abhängigkeiten und Anwendungsbausteine für das SDIM-Projekt bereitgestellt werden. Ebenso wird ein "sdim"-Benutzer angelegt, der die selbstgeschriebene Anwendung ausführen soll. Aus Gründen der Sicherheit muss das Ziel sein, das kein Anwendungsbestandteil unter dem "root"-Benutzer betrieben wird.
Damit die Integration von Dateien bei der Imageerstellung möglichst einfach gehalten wird, wird ein Overlay-Verzeichnis angelegt, welches alle benutzerspezifischen

Dateien beinhaltet. Der Inhalt dieses Verzeichnisses wird dann durch das Skript in das Image kopiert und steht so später auf dem Raspberry zur Verfügung.
Wenn das Skript erfolgreich ausgeführt wurde, erhält der Anwender eine Image-Datei, die über ein Werkzeug wie DD, Win32DiskImager oder Etcher auf eine SD-Karte geschrieben werden kann. Das Image soll beim Start direkt die Anwendung starten und für die Erstkonfiguration über das Webinterface vorbereiten.

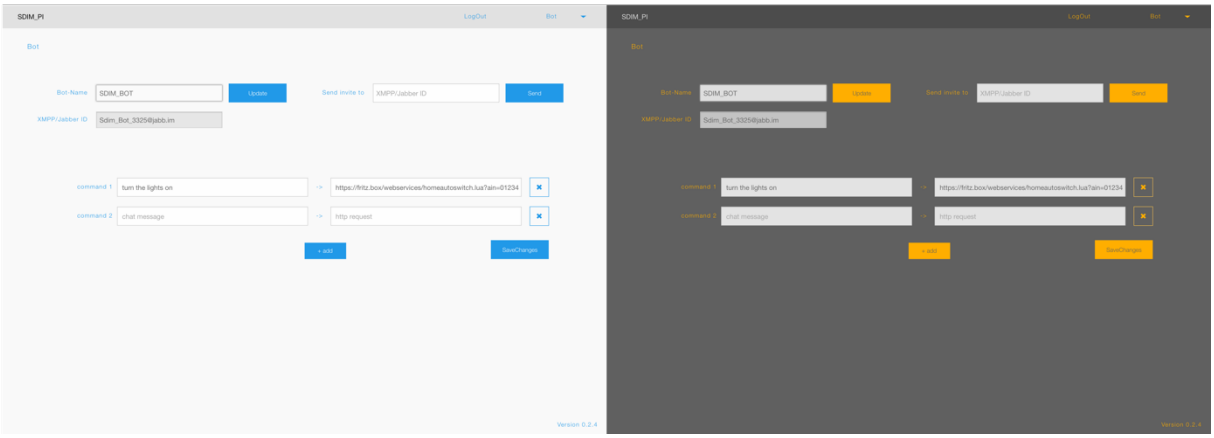
5. User Interface

Das User-Interface wird in Form einer Webanwendung realisiert, welche auf dem Raspberry Pi gehostet wird. Diese ist ausschließlich im lokalen Netz erreichbar. Erste Einstellungen am Pi werden nach dem initialen Start des Images in einem Einrichtungs-Wizard vorgenommen. Dieser besteht aus mehreren Elementen zum Setzen der Bezeichnung des Pi's, eines Passwortes und des Erscheinungsbildes (hell/dunkel), sowie zum Anzeigen der .onion-Adresse.

Ergänzt wird der Wizard um ein responsive Webinterface, über das die vorgenommenen Einstellungen nachträglich bearbeitet werden können. Außerdem sind weitere Funktionen, wie ein System-Log, eine Update-Funktion, eine Import-/Export-Funktion und Einstellungen bezüglich des angeschlossenen Displays möglich.



Neben den Einstellungen des XMPP-Servers, des Pi's und der Weboberfläche besteht die Möglichkeit, Einstellungen am SDIM-Bot vorzunehmen. So kann der Name des Bots festgelegt werden und es können Invites an die XMPP/Jabber-Profile der Nutzer versendet werden (um die Kommunikation mit dem Bot zu ermöglichen). Befehle können über eine Zuweisung mit http(s)-Requests verknüpft werden, um beispielsweise auf die Home-Automation der FRITZ!Box zuzugreifen.



Es ist zu erwähnen, dass die gezeigten Abbildungen lediglich einen ersten Entwurf darstellen. Im Laufe des Projektes kann es zu Abweichungen kommen. So sind auch nicht alle Funktionalitäten in den Mockups abgedeckt, wie z. B. eine Landingpage mit Login und eine Authentifikationsmöglichkeit mit der FRITZ!Box (die FRITZ!Box https-Requests erfordern eine gültige Session-ID).

6. Framework und Laufzeitumgebung (Frontend)

Es stehen viele Technologien zur Auswahl, mit denen die Webanwendung implementiert werden könnte. In einer ersten Planung wurden zwei Varianten gebildet. Die Wahl hängt von der Lauffähigkeit der Varianten auf dem Pi-System ab.

node.js + handlebars.js + Foundation + MongoDB

- NodeJS ist eine Laufzeitumgebung, die das client- und serverseitige Entwickeln mittels JavaScript erlaubt
- Handlebars dient zur Erstellung von HTML-Templates
- Foundation ist ein Framework zur Frontend-Gestaltung
- NodeJS ermöglicht das Anbinden einer dokumentenbasierten Datenbank wie mongoDB (dadurch werden Setting-Imports/Exports erleichtert, da die Daten bereits im JSON-Format vorliegen)

meteor.js + blaze.js + Foundation + MongoDB *(favorisiert)*

- MeteorJS basiert auf NodeJS und erleichtert die Konfiguration erheblich
- In Kombination mit der Blaze-Library wird Reaktivität der Oberfläche automatisch gewährleistet (Oberfläche bleibt mit Datenbasis synchron)
- Handlebars, sowie eine MongoDB-Instanz sind bereits angebunden

Eine MeteorJS Anwendung läuft im Kern in einer nodeJS Umgebung. Erweiterungen können wie bei nodeJS über npmJS-Pakete (https://www.npmjs.com/) hinzugefügt werden. Zusätzlich stehen in einer MeteorJS-Anwendung die AtmosphereJS-Pakete (https://atmospherejs.com/) zur Verfügung.

node.js	https://nodejs.org/en/
handlebars.js	http://handlebarsjs.com/
foundation.zurb.com	http://foundation.zurb.com/
mongoDB	https://www.mongodb.com/
meteor.js	https://www.meteor.com/
blaze.js	http://blazejs.org/

7. Zusätzliche Funktionalität

Um die Chat-Funktion, für die der XMPP-Server eigentlich dient, erweitern zu können, soll ein Chat-Bot implementiert werden. Dieser soll über verschiedenste individualisierbare Befehle ansprechbar sein. Vorstellbar sind unteranderem die Interaktion mit Smart-Home Geräten und einfache Befehle zur Informationsabfrage.

Eine der größeren Herausforderungen ist es, die Befehle aus natürlicher Sprache zu extrahieren. Da dies unter Umständen schnell in die Erstellung einer Art KI ausarten kann, soll sich zunächst auf einfache Befehle beschränkt werden. Diese Befehle können mittels einfachen Schlüsselworten (z. B. „!Hell“) übermittelt werden, bis hin zu simplen Sätzen, aus denen die Schlüsselwörter extrahiert werden.
Der Bot hängt von der gewählten Server- und der gewählten Laufzeitumgebung ab. Damit der Bot jederzeit erreichbar ist, muss dieser im System dauerhaft hinterlegt sein. Hierfür erhält er eine eigene XMPP/Jabber-ID.

Auf folgender Website ist eine Liste mit verschiedenen Chat-Bots zu finden. Einige dieser sollen als Vorbild dienen.
http://meta-guide.com/bots-agents-assistants/chatbots/jabber-xmpp-chatbots