

HÖHERE TECHNISCHE BUNDESLEHRANSTALT
KLAGENFURT, MÖSSINGERSTRASSE

ABTEILUNG ELEKTRONIK UND TECHNISCHE
INFORMATIK

MINIPROJEKT

Infrastruktur des Rendezvous-Servers für DA

eingereicht von Quentin Wendegass
 Lukas Kuster

Projektbetreuer Dipl.-Ing. Robert Oyrer

Diese Diplomarbeit entspricht den Standards gemäß dem Leitfaden zur Umsetzung der Reife- und Diplomprüfung des BM|BF in der letztgültigen Fassung.

Klagenfurt, am 07.06.2018

Kurzbeschreibung zur Diplomarbeit

Bei unserem Miniprojekt handelt es sich um einen Teil unserer zukünftigen Diplomarbeit. Hierbei soll ein Gerät entwickelt werden, welches mit einer SIM-Karte ausgestattet ist und im Mobilfunknetz eingeloggt ist, außerdem ist dieses auch noch mit dem Internet über ein lokales Netzwerk verbunden (entweder über WiFi oder Ethernet). Mittels Client-Apps welche für Desktop (Mac, Windows, Linux), sowie Mobil-Geräte (vorerst nur iOS) entwickelt werden, kann man sich dann mit diesem Gerät verbinden. Dies ermöglicht dem Nutzer über die im Gerät befindliche SIM-Karte Anrufe zu tätigen/empfangen, sowie Nachrichten zu versenden/empfangen. Dabei muss dieser mit dem Client-Gerät nur eine aufrechte stabile Internet-Verbindung in einem beliebigen Netzwerk weltweit haben. Dies ermöglicht es zum Beispiel Roaming-Gebühren auf Auslandsreisen zu umgehen, was vor allem für Vielreisende interessant. Dabei ist die Telefonnummer durchgehend erreichbar, sollte über keine Client-Anwendung angenommen werden, wird auf eine virtuelle Sprachbox weitergeleitet welche anschließend über alle Client-Anwendungen erreichbar ist.

Inhaltsverzeichnis

1. Aufgabenstellung	4
2. Projektmanagement	4
2.1. Produktstrukturplan	4
2.2. Projektstrukturplan	5
2.3. GANTT-Diagramm	6
2.4. Arbeitspakete	6
3. Hole – Punching	7
3.1. Allgemein	7
3.2. Anwendung	7
4. Software	8
4.1. Client	8
4.2. Server	9
4.3. API	10
4.4. Visualisierung	11
5. Testergebnisse	13
6. Fazit	14

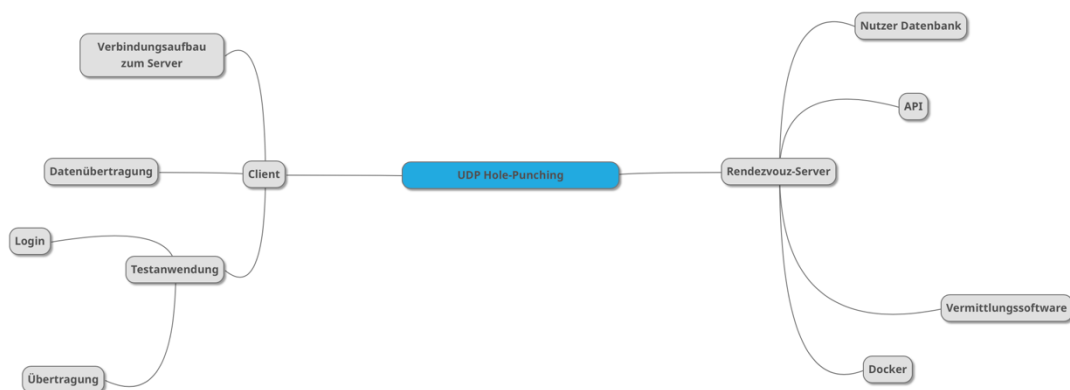
1. Aufgabenstellung

Als Miniprojekt behandeln wir den Verbindungsaufbau zwischen dem Client und der SIM-Box (nachfolgend als Device bezeichnet). Dabei stellt sich das technische Problem das die beiden Geräte in zwei unterschiedlichen privaten Netzwerken hinter zwei verschiedenen NATs liegen. Dafür nutzen wir sogenanntes "hole-punching", dabei werden die beiden Geräte über einen Server (von uns Rendezvous-Server genannt) vermittelt. Danach kann eine direkte Verbindung zwischen den beiden Geräten herstellen und Sprachdaten, sowie andere Informationen, direkt übertragen. Dies erspart uns hohen Datenfluss über den dritten öffentlich-zugänglichen Server, da nur die Vermittlung und nicht die komplette Übertragung über diesen erfolgt.

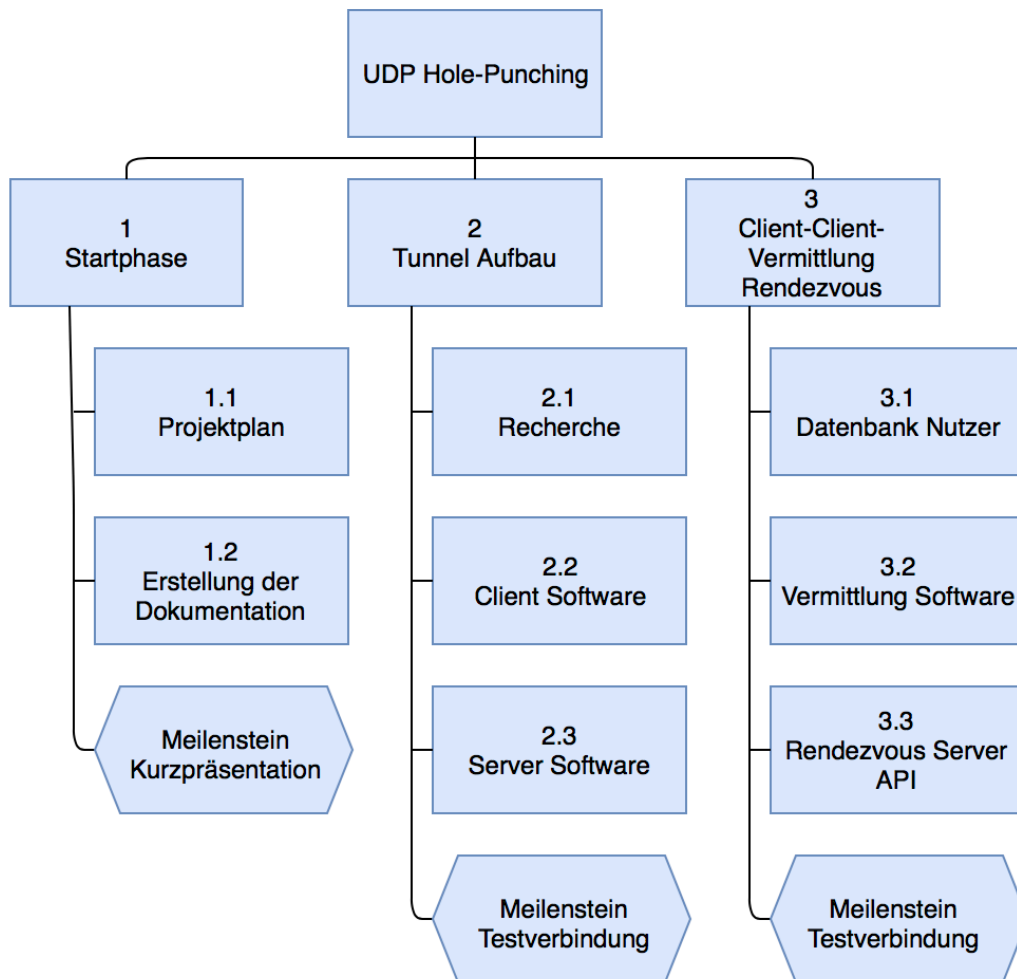
Grundsätzlich teilen wir das Miniprojekt in zwei Teile: die Tunnelaufbau-Phase und die Client-Device-Vermittlungs-Phase. In der ersten Phase werden wir versuchen eine einfache Verbindung zwischen zwei Geräten in zwei verschiedenen getrennten Netzwerken über "UDP hole-punching" herzustellen. Dies erfolgt Client-, wie auch Serverseitig, in Python. Danach wird dieses Programm erweitert, das es mehrere Nutzer, darunter mehrere SIM-Geräte sowie mehrere Clients, gleichzeitig unterstützt. Dadurch wird die serverseitige Infrastruktur des Rendezvous-Servers für die Diplomarbeit geschaffen.

2. Projektmanagement

2.1. Produktstrukturplan

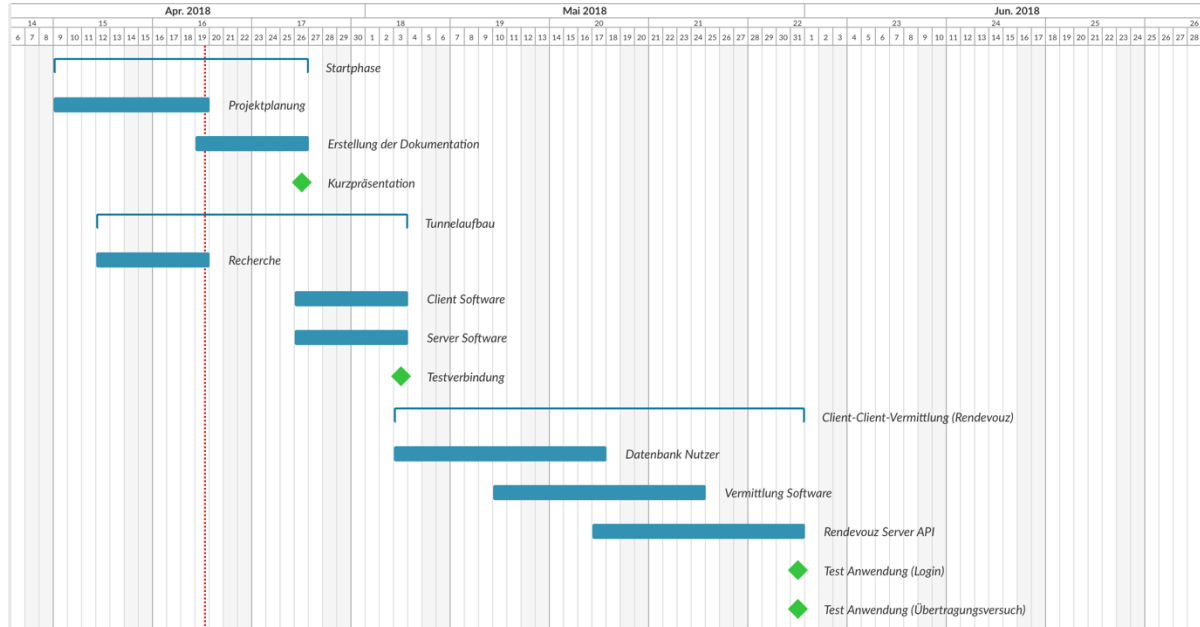


2.2. Projektstrukturplan



2.3. GANTT-Diagramm

Miniprojekt - Wendegass/Kuster - UDP Hole-Punching/Vermittlung



2.4. Arbeitspakete

Arbeitspaket	Beschreibung, Ziel, Arbeitsinhalt	Dauer/h	Ressourcen Schüler
1.1	Projektplanung	4	W + K
1.2	Erstellung der Dokumentation	4	W + K
2.1	Recherche	5	W + K
2.2	Client Software	6	W
2.3	Server Software	6	K
3.1	Datenbank Nutzer	6	K
3.2	Vermittlung Software	6	W + K
3.3	Rendezvous Server API	6	W

3. Hole – Punching

3.1. Allgemein

Hole-Punching beschreibt das Verfahren, trotz Firewall eine Kommunikation zwischen zwei Teilnehmern, welche sich jeweils hinter einem NAT (Network Address Translator) befinden, zu erlauben. Normalerweise verwendet man in diesem Fall einen öffentlich zugänglichen Server über den jegliche Daten, die zwischen den Geräten übertragen werden sollen, geschickt werden. Dieser wird sozusagen als „Mittelsmann“ eingesetzt. Da bei dieser Verfahrensweise aber jede Kommunikation über den Server geleitet wird, werden diverse Kosten für Verbindungsaufbau, Datenübertragung, -verarbeitung, usw. fällig. Aufgrund dessen und um eine spätere End-zu-Endverschlüsselung zu realisieren, haben wir uns entschieden eine direkte Verbindung zwischen den Teilnehmern aufzubauen.

Da bei einem Übertragungsversuch zwischen zwei unbekannten Geräten, welche sich in getrennten Netzwerken befinden, eine Firewall (bzw. NAT) befindet, ist es nicht möglich eine Verbindung aufzubauen. Deswegen verwenden wir das Verfahren „Hole-Punching“^[1]. Bei diesem wird zwar auch ein öffentlicher Server benötigt, aber dieser wird nicht für die eigentliche Übertragung der Daten, sondern nur für die Verbindung der beiden Teilnehmer benötigt.

Um eine Verbindung zwischen zwei Teilnehmern herzustellen, müssen sich alle Teilnehmer (Clients und Devices) beim Server anmelden. Der Server weist dann die Clients und Devices zueinander zu und schickt jedem die IP-Adresse und den geöffneten Port am anderen Teilnehmer. Danach können diese miteinander kommunizieren, da beide einen Port für den Server geöffnet haben und die IP-Adresse und den Port des anderen kennen.

3.2. Anwendung

Für unser Projekt haben wir ein einfaches Chatprogramm mit diesem Verfahren realisiert. Es kann sich ein Client mit einem Nutzernamen und Password beim Server anmelden. Dieser vergleicht diesen über unsere API mit der Datenbank, um zu bestätigen dass der Nutzer registriert ist. Wenn sich zwei Geräte mit dem gleichen Nutzer anmelden, werden diese dann miteinander verbunden und können sich gegenseitig direkt Nachrichten schicken, ohne dass der Datenverkehr über einen Server geleitet wird.

4. Software

Für die Serverseitige Vermittlungssoftware und das Client-Programm, haben wir uns entschieden, die Programmiersprache Python zu verwenden. Python ist für so ein Vorhaben eine gute Wahl, wegen der einfachen Syntax und der ausgiebigen Standardbibliotheken.

4.1. Client

Das Client-Programm ist relativ simpel zu Verstehen. Erst wird eine Nachricht mit dem Nutzernamen und Password an den öffentlichen Server gesendet. Wenn zwei Clients sich mit dem gleichen Nutzer anmelden, empfängt der Client die IP-Adresse und den Port des jeweils anderen. Danach werden zwei Threads gestartet, um das gleichzeitige empfangen und schicken von Nachrichten zu ermöglichen. Der Thread der für das Senden zuständig ist, schickt dann einen Text, der in der Konsole eingegeben wurde an die erhaltene Adresse vom anderen Client.

```
import socket
import sys
import thread
from util import *

HOST = '182.10.10.13'
PORT = 9999

def send(sock, addr):
    print(addr)
    while True:
        data = raw_input()
        sock.sendto(data, addr)

def receive(sock):
    while True:
        data, addr = sock.recvfrom(1024)
        print('client received: {} {}'.format(addr, data))

def main(username, password):
    sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
    sock.sendto(username + ";" + password, (HOST, PORT))

    data, addr = sock.recvfrom(1024)
    addr = msg_to_addr(data)

    thread.start_new_thread(send, (sock, addr))
    thread.start_new_thread(receive(sock), ())

if __name__ == '__main__':
    main(sys.argv[1], sys.argv[2])
```


4.2. Server

Der Server wartet, bis sich ein Client versucht zu verbinden. Wenn ein Client eine Anfrage an den Server schickt, nimmt dieser den Nutzernamen und Password und bestätigt mit unserer API, dass der Nutzer existiert und kein falsches Password eingegeben wurde. Wenn dies zutrifft, speichert er den Nutzer in einer Liste ab. Wenn sich dann der gleiche Nutzer von einem anderen Gerät verbindet, vergleicht der Server ob sich ein anderer Client bereits mit diesem Nutzer angemeldet hat und bei Erfolg schickt er den beiden Clients die Adresse des jeweils anderen.

```
import logging
import socket
import sys
from util import *
import requests

logger = logging.getLogger()
users = []

class User:
    def __init__(self, username, password, address):
        self.username = username
        self.password = password
        self.address = address

    def __eq__(self, other):
        if type(other) == type(self):
            return True if other.username == self.username and other.password == self.password else False

    def __repr__(self):
        return "User(username: {}, password: {}, Address: {})"\
            .format(self.username, self.password, addr_to_msg(self.address))

def main(host='10.68.1.44', port=9999):
    sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
    sock.bind((host, port))
    while True:
        data, addr = sock.recvfrom(1024)
        username, password = str(data).split(";", 1)
        logger.info("Username: %s, Password: %s", username, password)

        logger.info("connection from: %s", addr)
        data = {'username': username,
                'password': password}
        response = requests.post("http://127.0.0.1:8080/api.php", data=data)
        if response.status_code == 200:
            user = User(username, password, addr)
            if user in users:
                index = users.index(user)
                logger.info("server - send client info to: %s", users[index])
                sock.sendto(addr_to_msg(user.address), users[index].address)
                logger.info("server - send client info to: %s", users[index])
                sock.sendto(addr_to_msg(users[index].address), user.address)
            else:
                users.append(user)

if __name__ == '__main__':
    logging.basicConfig(level=logging.INFO, format='%(asctime)s - %(message)s')
    main(*addr_from_args(sys.argv))
```

4.3. API

Die API wurde einfachheitshalber in PHP umgesetzt. Die einzige Funktion der API, ist die Verifizierung des Nutzers. Als Argument muss ein Nutzernamen und ein Passwort an den Endpunkt geschickt werden. Erst wird eine Verbindung zu unserer Datenbank „miniproject“ hergestellt. Dann werden alle Spalten zurückgegeben, bei denen der Nutzernamen und das Passwort mit den übergebenen Argumenten übereinstimmen. Wenn keine Spalte zurückgegeben wurde, bedeutet das entweder, dass der Nutzer nicht vorhanden ist, oder das, das Passwort nicht übereinstimmt. Bei diesem Szenario wird ein 403 Status-Code zurückgegeben. Wenn der Nutzer existiert und das Passwort stimmt, wird ein 200 Status-Code zurückgegeben.

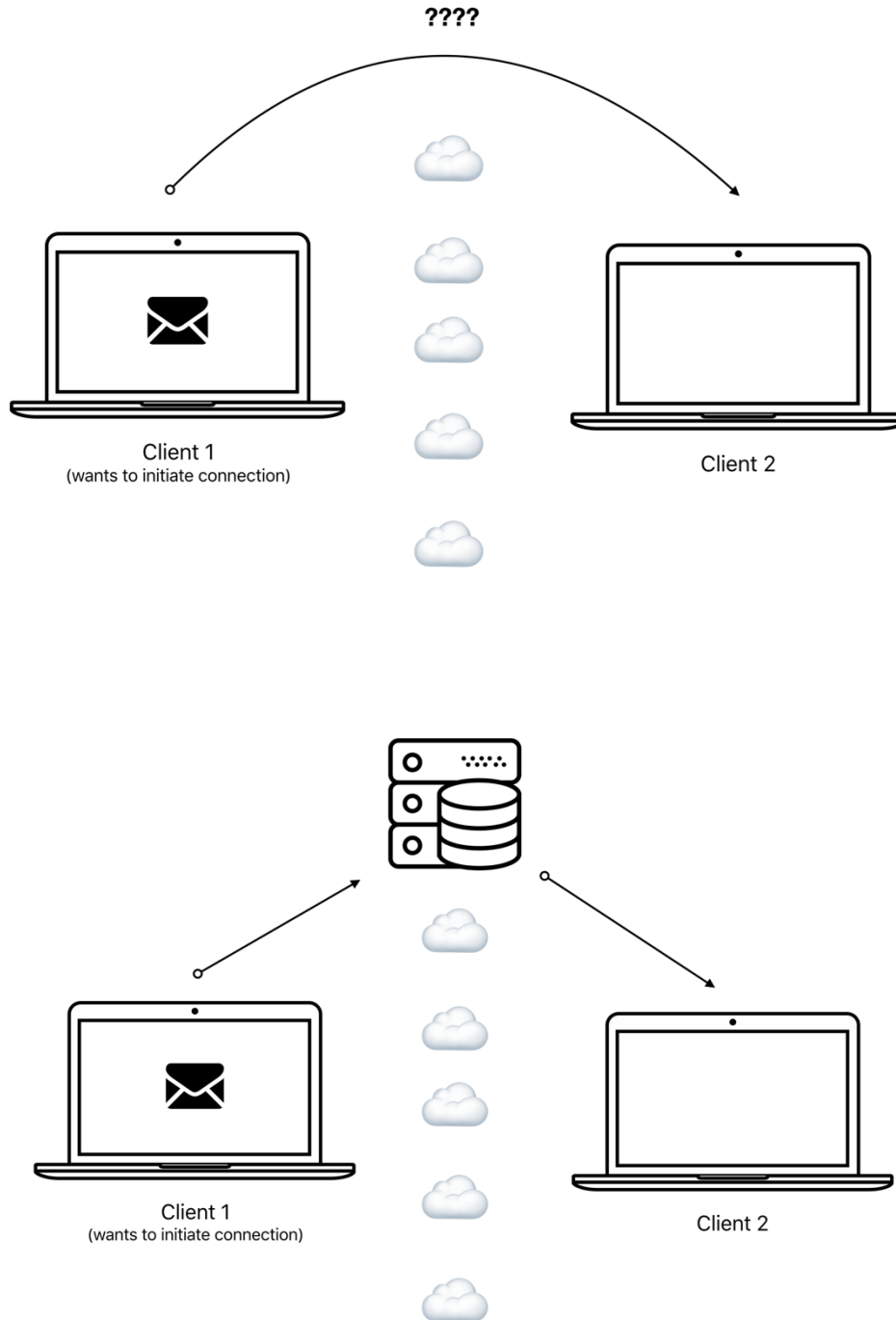
```
<?
$username = $_POST["username"];
$password = $_POST["password"];

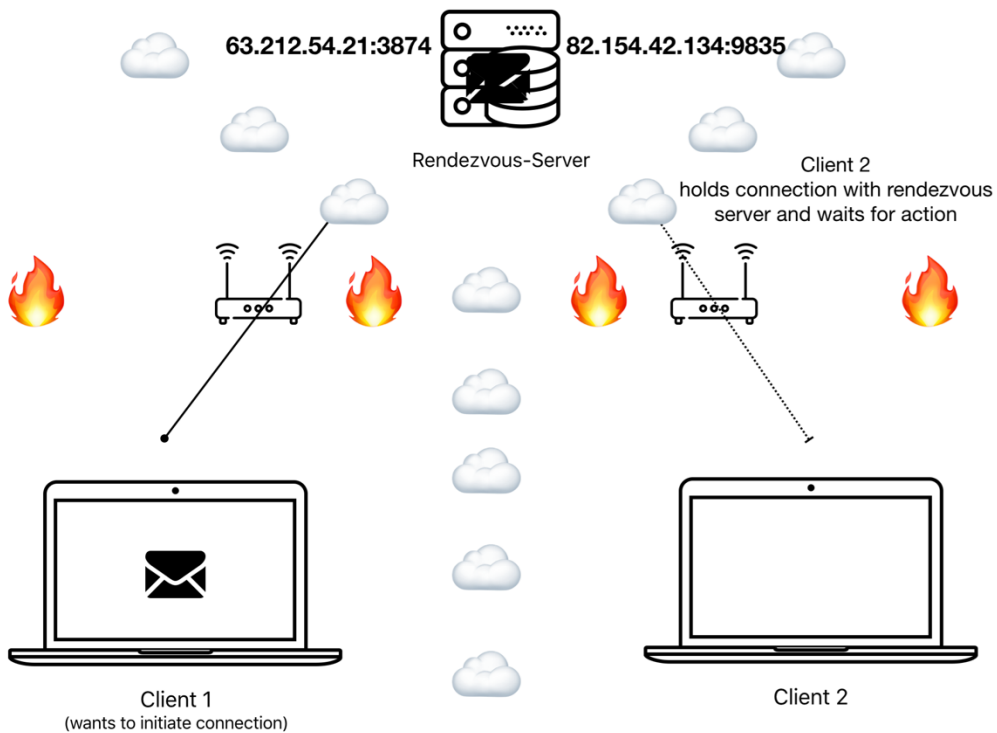
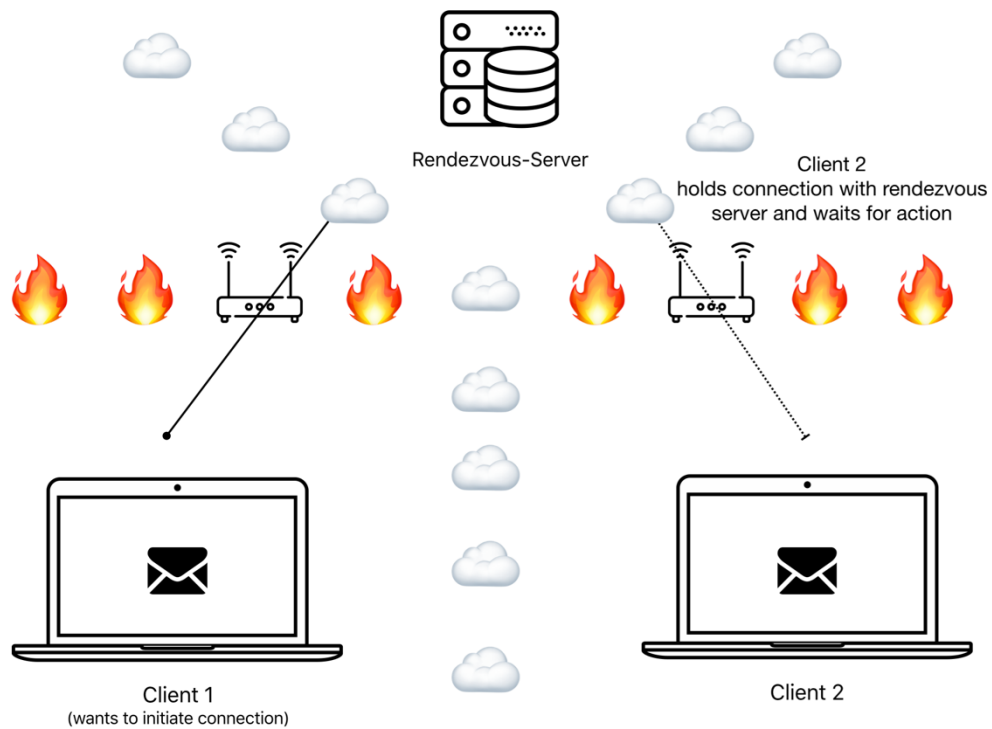
require('db.php');
$conn->select_db( dbname: "miniproject");
$conn->set_charset( charset: "utf8");

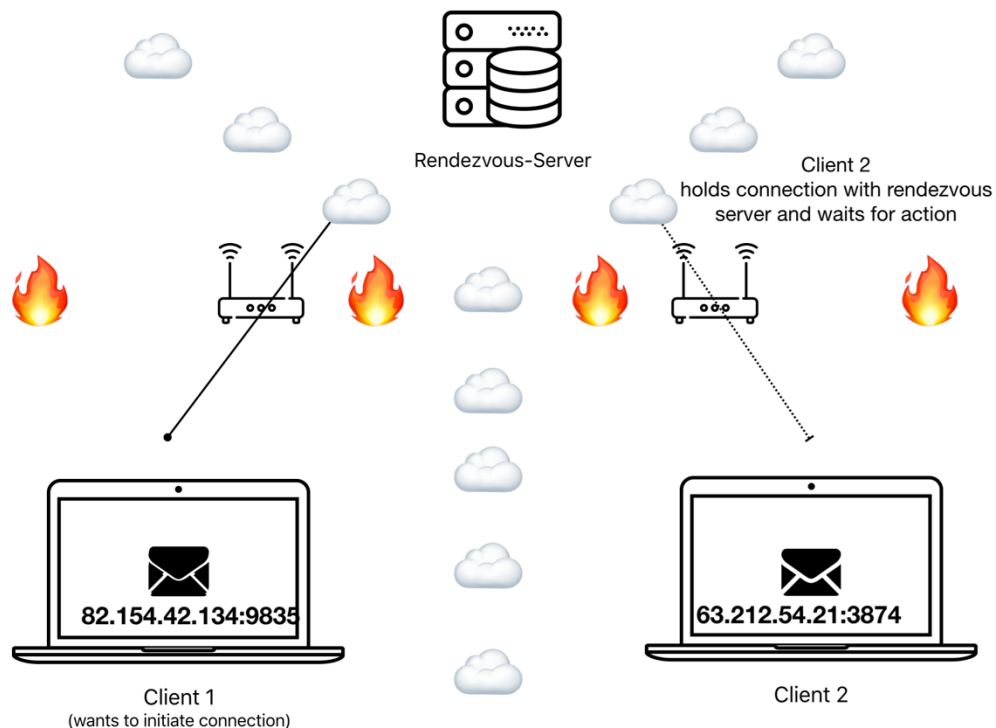
$sql = "SELECT uid, password FROM `users` WHERE `username` = '". $username. "' AND `password` = '". $password. "'";

if ($result = $conn->query($sql)) {
    if($result->num_rows != 1) {
        http_response_code( response_code: 403);
        $status["status"] = 403;
        echo json_encode($status);
    }else{
        http_response_code( response_code: 200);
        $status["status"] = 200;
        echo json_encode($status);
    }
}
}
}
?>
```

4.4. Visualisierung







5. Testergebnisse

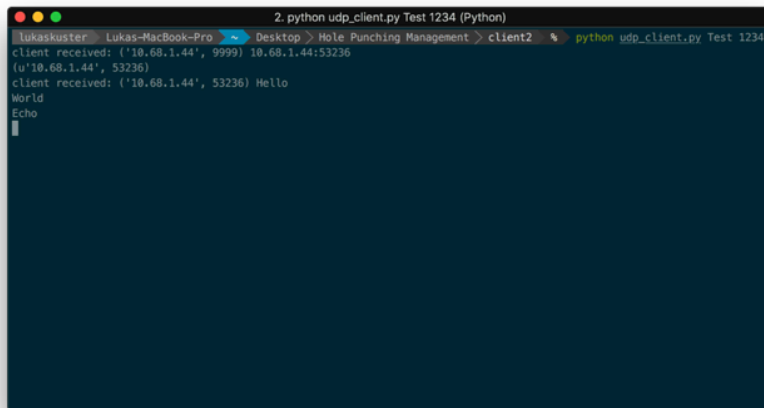
Rendezvous-Server

```
2018-05-28 09:58:42,601 - Username: Test, Password: 1234
2018-05-28 09:58:42,601 - connection from: ('10.66.117.168', 53862)
2018-05-28 09:58:49,066 - Username: Test, Password: 1234
2018-05-28 09:58:49,066 - connection from: ('10.66.1.44', 59366)
2018-05-28 09:58:49,066 - server - send client info to: User(Username: Test, Password: 1234, Address: 10.66.117.168:53862)
2018-05-28 09:58:49,066 - server - send client info to: User(Username: Test, Password: 1234, Address: 10.66.117.168:53862)
```

Client (Quentin's Mac – Client der sich mit Device verbindet)

```
quentinwendegass:python-nat-hole-punching$ python2.7 udp_client.py Te
st 1234
client received: ('185.69.244.13', 9999) 10.66.117.168:56254
(u'10.66.117.168', 56254)
Hello
client received: ('10.66.117.168', 56254) World
client received: ('10.66.117.168', 56254) Echo
```

Device (Lukas' Mac – Device das auf Client-Verbindung wartet)



```

2. python udp_client.py Test 1234 (Python)
lukaskuster@Lukas-MacBook-Pro: Desktop > Hole Punching Management > client2 % python udp_client.py Test 1234
client received: ('10.68.1.44', 9999) 10.68.1.44:53236
(u'10.68.1.44', 53236)
client received: ('10.68.1.44', 53236) Hello
World
Echo
  
```

Es konnte eine Testverbindung zwischen zwei Geräten hergestellt werden, welche sich in zwei verschiedenen getrennten Netzwerken befanden (Client -> Schulnetzwerk, Device -> Hotspot von Mobilgerät). Diese Verbindung konnte ohne Probleme hergestellt werden. Eine Live-Demonstration erfolgte auch während der Miniprojekt Präsentation. Durch Informationen die wir aus einem Whitepaper^[1] zu „UDP hole-punching“ beziehen, zeigen Tests in verschiedenen Netzwerken das 80% der Verbindungen erfolgreich sind. Daher sollte auch eine so genannte „Fallback“-Variante realisiert werden, bei welchen die Daten normal über den Vermittlungsserver (als Mittelsmann) geschickt werden. So entstehen zwar Kosten der Übertragung, diese können aber durch die Nutzung des „hole punchings“ um rund 80% reduziert werden.

6. Fazit

Es konnte eine relativ stabile Verbindung zwischen mehreren Clients und einem Device hergestellt werden. Um das ganze Marktreif bzw. für unsere Diplomarbeit einsetzbar zu machen sind noch weitere Tests nötig. Außerdem sind wir während der Entwicklung auf weitere schon existierende und getestete Protokolle bzw. Möglichkeiten gestoßen. So wurde ich (Lukas Kuster) unter anderem von einem Apple FaceTime Engineer auf das WebRTC-Protokoll hingewiesen mit welchen wir in der Lage sind, die selbe womöglich auch bessere Verbindung herzustellen. Mit diesem Miniprojekt konnten wir aber Einsicht in den fundamentalen Aufbau solcher Protokolle nehmen. Für die Diplomarbeit werden wir aber auf ein schon existierendes und bewährtes Protokoll bzw. Ansatz nehmen. Und nicht das Rad komplett neu erfinden.

[1] Peer-to-Peer Communication Across Network Address Translators, Bryan Ford (MIT), Pyda Srisuresh (Caymas Systems, Inc.), Dan Kegel (found at <http://www.brynosaurus.com/pub/net/p2pnat/>).