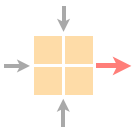
ETH zürich



Networked Systems
ETH Zürich — seit 2015

VPN for Isolated Docker Networks
An Implementation for the Mini-Internet Platform

Bachelor Thesis

Author: Leonard Wechsler

Tutor: Lukas Röllin

Supervisor: Prof. Dr. Laurent Vanbever

November 2024 to February 2025

Acknowledgements

I would like to express my sincere gratitude to Prof. Dr. Laurent Vanbever, Lukas Röllin and the Networked Systems Group for giving me the opportunity to work on this Bachelor Thesis. Their guidance and support during this project has been invaluable. I am thankful to my family for their emotional and financial support, which made this journey possible. Last but not least, I would like to thank my friends and roomates for their encouragement and feedback.

Abstract

Understanding how the internet works is essential for many engineers and researchers. In the course "Communication Networks" students study this topic in an interactive fashion. During one semester they build an isolated internet-like network called "Mini-Internet". Although this is a great approach to see the workings from a network managers perspective, there is a minor caviat. The Mini-Internet does not have the ability to connect personal devices to the network. In this thesis the implementation of a Virtual Private Network (VPN) will be discussed. The VPN allows students to connect their own devices to the Mini-Internet. In this way, the Mini-Internet will feel a little more like a "real" network.

This work is tailored to, but not limited to, the Mini-Internet. It applies everywhere, where easy and secure access to an isolated Docker network is required.

Contents

Chapter 1

Introduction

1.1 Motivation

ETH's Networked Systems Group (NSG) has developed the "Mini-Internet" platform. This platform simulates an interconnected network. Docker containers emulate various components like switches, routers and hosts. The Mini-Internet is isolated from the real internet.

In the "Communication Networks" course students build a network using the Mini-Internet platform. To make the course lab more interactive, an idea is to connect to the Mini-Internet with devices other than the simulated ones. It would be nice if students could connect their laptops and smartphones and send traffic over the Mini-Internet.

1.2 Task and Goals

The technical goal of the project is to connect personal devices to the Mini-Internet. In order to achieve this, a Virtual Private Network (VPN) is built using the Wireguard protocol. The VPN connects a client to the Mini-Internet, using a tunnel over the internet. This way, users can remotely access the Mini-Internet as if they were physically connected.
The connection process should be easy and intuitive. The implementation should be lightweight, robust and easy to manage.

Steps to achieve this goal:

1. Build a Wireguard connection between client and Mini-Internet
2. Automate the setup for larger scale networks
3. Create a user interface to manage the Wireguard connections

1.3 Overview

To understand this thesis, chapter 2 outlines necessary background knowledge. This includes information on the Mini-Internet, the Wireguard protocol and the webserver used for the user interface. The implementation is described in Section 3. In chapter 4, a performance analysis is conducted. The thesis is concluded with a brief outlook on future work (chapter 5) and a summary (chapter 6).

Chapter 2

Background and Related Work

2.1 Mini-Internet

The "Communication Networks" lecture of ETH's Networked Systems Group (NSG) [1] teaches students how the internet works. In the lab, students build a "Mini-Internet" where each group of students manages their own Autonomous System (AS). These ASes then form an interconnected network. The Mini-Internet is built using a platform developed and maintained by the NSG [2][3].

At the time, the platform runs on a server as a collection of Docker containers. The container's network is separated from the internet, but allows access via SSH and a public website.

As described in 1.2, the goal of this thesis is to implement a VPN for the Mini-Internet. In an earlier version a VPN implementation for the Mini-Internet did already exist. However, this feature was discontinued when migrating the Mini-Internet from a VM-based to the Docker-based approach. Historically, the VPN was difficult to configure and to use. Additionally, it was never tested with other Operating Systems than Linux/Ubuntu.

2.2 Wireguard VPN

A Virtual Private Network (VPN) virtually extends a private network (i.e. the Mini-Internet) over another network (i.e. the internet). This keeps the private network isolated, but makes it accessible over a (encrypted) tunnel [4].
For the implementation of the VPN, it was decided to use "Wireguard" - a lightweight, open-source and easy to manage protocol [5][6]. Wireguard works by connecting "Peers" or "Clients" over an encrypted UDP tunnel. On each client Wireguard creates a network interface that routes traffic through the tunnel.

2.3 Docker and Namespaces

The implementation requires some understanding of networking in Docker. There are several networking modes available. For this thesis it is sufficient to understand the bridge network mode [7]. In this mode Docker connects the network interface of a container to other containers in the same (user-specified) network. On a Kernel level Docker uses Linux network namespaces to isolate the container's network stack from the host's network stack.

2.4 User Interface and Webserver

The Mini-Internet has a user interface that can be accessed over a webserver. In this case, the weberver runs as a Docker container and is powered by Flask [8]. Because of this containerization, the webserver can not just access the router containers. Section 3.4 explains how observer processes are used to get data from the routers. These observers use supervisord [9] as a process control system.

Chapter 3

Design

This chapter contains the design of the project. We begin by discussing the implementation for a single Docker container. Then, we explain how to integrate the VPN into multiple containers of the Mini-Internet. Finally, we discuss rate limiting and the user interface.

3.1 The Wireguard Tunnel

The goal is to establish a VPN connection to a Docker container. As explained Section 2.2, Wireguard uses a tunnel to connect peers over the internet. However, the Docker container does not have access to the internet.
To solve this issue, one would typically use Dockers port publishing capabilities. Because the Docker network is internal, it would require to set up an additional bridge network. However, this would add unnecessary complexity and would require sophisticated routing tables to avoid complications with the Mini-Internet's routing. Additionally, port publishing can only be configured when building the container and not during runtime. It can not be modified when restarting a container - the container has to be replaced by a new one.

Wireguard provides a more efficient solution for this problem: When a Wireguard interface is created, it is bound to a socket. If the interface is moved into the container's network namespace, the socket descriptor stays in the original namespace [10]. Therefore, the socket uses the host's network stack and not the one of the container. This way, the encrypted Wireguard packets can reach the internet, even though the Wireguard interface is inside the Docker container. This solution does not require Docker's port publishing option and can be configured without restarting the container. A schematic of set up is shown in Figure 3.1.
Depending on the hosts firewall configuration, it could be necessary to allow traffic to the Wireguard ports. Note that it is is not necessary to publish one port for every Wireguard interface. It is possible to use a tool like iptables to route the traffic to the different interfaces with only one external connection. However this option will not be discussed here (See Chapter 5).
Listing 3.1 shows the implementation in bash.

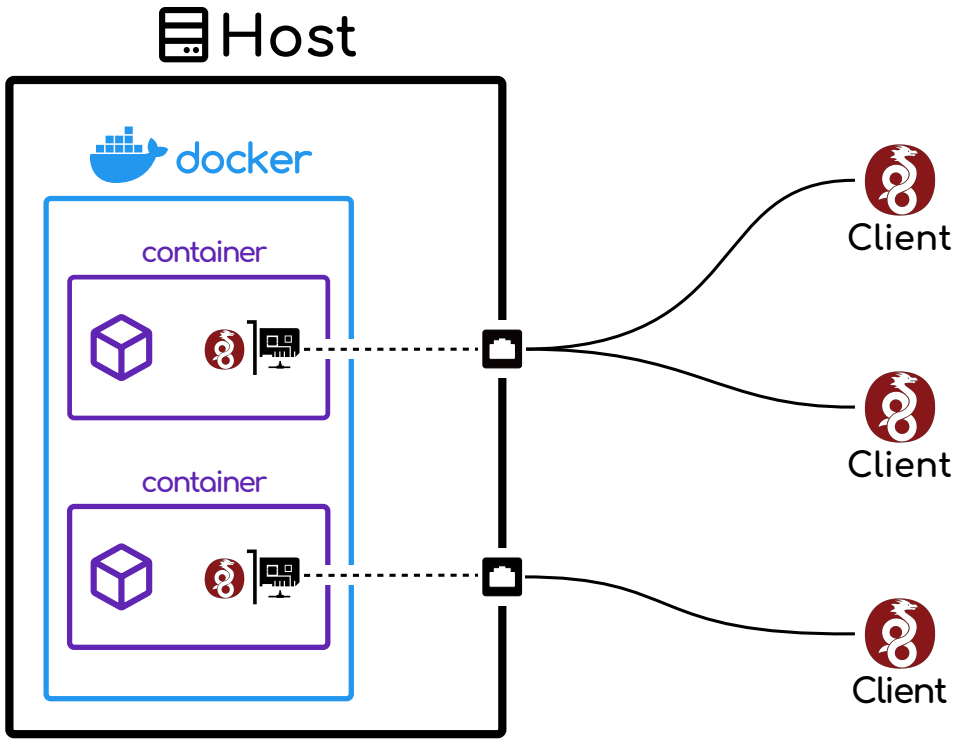


Figure 3.1: Wireguard tunnel overcomes container isolation

```
# Create Wireguard interface
ip link add vpn type wireguard

# Move interface to the container. The container's ${PID} can be acquired by
    using docker inspect.
ip link set vpn netns "${PID}"

# Configure IP address
nsenter --net=/proc/"${PID}"/ns/net ip address add "${ip_address}" dev vpn

  # Set a Wireguard configuration, which must be placed at this path inside the
      container:
docker exec -u root "${container_name}" wg setconf vpn /etc/wireguard/interface.
    conf

  # Enable interface
nsenter --net=/proc/"${PID}"/ns/net ip link set vpn up

# Set firewall exception
ufw allow "${listen_port}" > /dev/null
```

Listing 3.1: Create a Wireguard Interface and Move it into a Container

3.2 Integration into the Mini-Internet

There are several possible approaches to include the VPN into the Mini-Internet:

- Run Wireguard on an existing router.
- Run Wireguard on an existing host.
- Create a dedicated container for each Wireguard connection.
- Create one Wireguard proxy container that handles all connections.

For our purposes, the best solution is to run Wireguard on an existing router. It is the simplest and most straightforward option.
Creating a separate container for each Wireguard interface would use additional resources such as CPU time and memory. Using a proxy container would require sophisticated routing tables for packets to reach the correct AS. Running Wireguard on a host container leads to a scenario where a host needs to act as router for the VPN packets. An already existing router provides the necessery framework to easily route the packets.
When Wireguard is run on a router, it seems like the client is physically connected to an interface of the router. Each AS can run Wireguard on one or multiple routers. The decision was made to run Wireguard on all routers. This allows students to connect their devices to every single router in their group's AS.

After creating a Wireguard interface as described in Section 3.1, some additional steps are required. First, an IP prefix is assigned to the interface. As each AS has a subnet X.0.0.0/8 we can allocate a prefix in this range for the Wireguard connections. For example, each interface can be assigned a VPN subnet with the IP range X.(50 + Y).0.0/24, where X is the group number and Y the router number. Second, the router needs to include prefix in the OSPF configuration. Inside the container we execute:

```
vtysh -c "conf t" -c "router ospf" -c "network ${interface_ip} area 0"
```

The VPN is integrated into the startup sequence. A startup script loops through every router and creates a Wireguard interface and peer configuration files. Although it is possible to add more peers during runtime, we decided on a fixed number of peers. This is sufficient for the Mini-Internet, as only a limited amount of students per group use the VPN. In return, the set up and configuartion is simpler. The number of peers per interface can be set in the configuration file. The user interface (Section 3.4) is set up by generating usernames and passwords and starting the observer processes. In the end, a cleanup script removes the interfaces, stops the observers and disables the VPN webpage. The startup and cleanup scripts do not interfere with a running Mini-Internet instance. It is possible to indepentently deactivate the VPN and restart it with a new configuration.

3.3 Rate Limits

Peers could potentially overwhelm the Mini-Internet with excessive network traffic. There are two possible options to limit the traffic volume in the Mini-Internet:

1. We can limit the bandwidth of the links. This method restricts how much traffic can be sent between two neighboring routers.

2. We can rate limit the Wireguard interface. This method restricts how much traffic is allowed to enter the Mini-Internet over the VPN connection.

The first option is already implemented. The maximal bandwith can be defined when setting up the virtual links between two routers. Listing 3.2 shows the code for implementing a simple rate limit for a Wireguard interface that is inside a container.
The effects of these limits is analysed in chapter 4.2.

```
1    nsenter --net=/proc/"${PID}"/ns/net tc qdisc add dev vpn root tbf rate ${
         VPN_LIMIT_RATE} burst ${VPN_LIMIT_BURST} latency ${VPN_LIMIT_LATENCY}
```

Listing 3.2: Rate limiting a Wireguard Interface

3.4 Webserver and User Interface

Establishing a connection to the Mini-Internet should be as easy as possible. The Mini-Internet already includes a webserver with an UI for the students. For the VPN, a new page is added to the webserver. To access this page, students must sign in with a password. On this page, all available Wireguard interfaces are listed. From this list, the student selects a router to connect to. To connect to Wireguard, the student needs to download the Wireguard app from the official website. Finally, pairing is as easy as downloading the config file or scanning a QR-Code. The UI

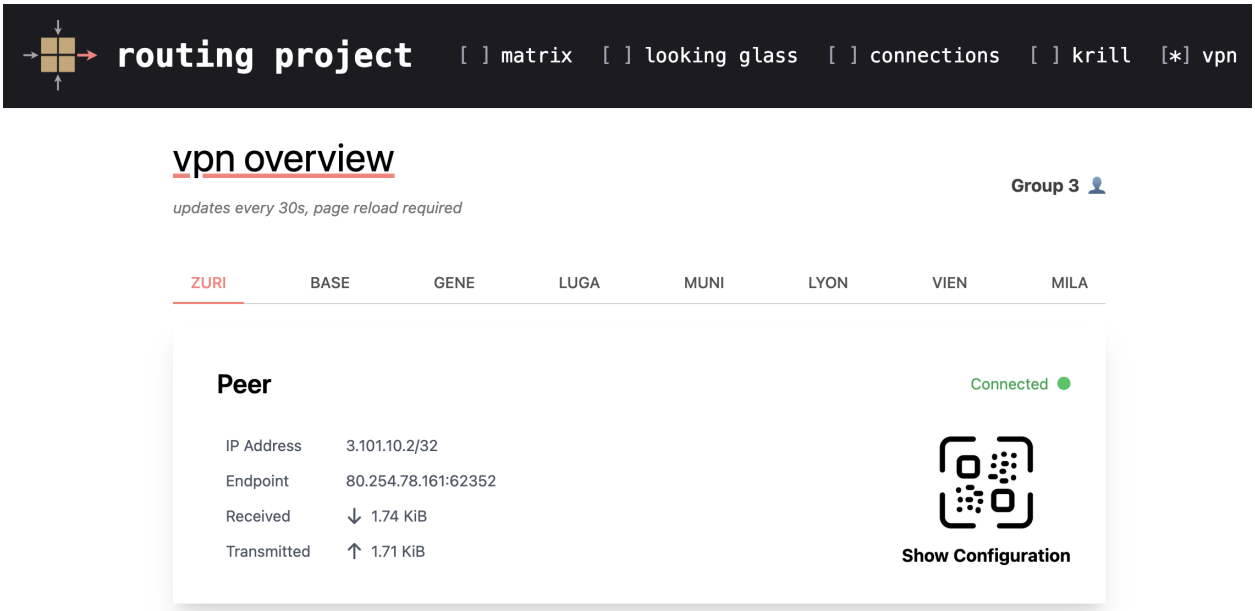


Figure 3.2: Screenshot of the User Interface

does not only simplify the connection process - it also provides information on active Wireguard connections. As seen in Figure 3.2, the client's connection status, IP address, and the amount of traffic sent and received is displayed.

The backend software of the UI is illustrated in Figure 3.3. Because of the containerization, the

webserver can not directly access a router. Therefore, each router runs an observer process. The observer periodically reads out the current state of the interface. This information is stored outside of the container. The webserver parses the observer's output and stores the information in a database. Another script accesses the database and generates the website.

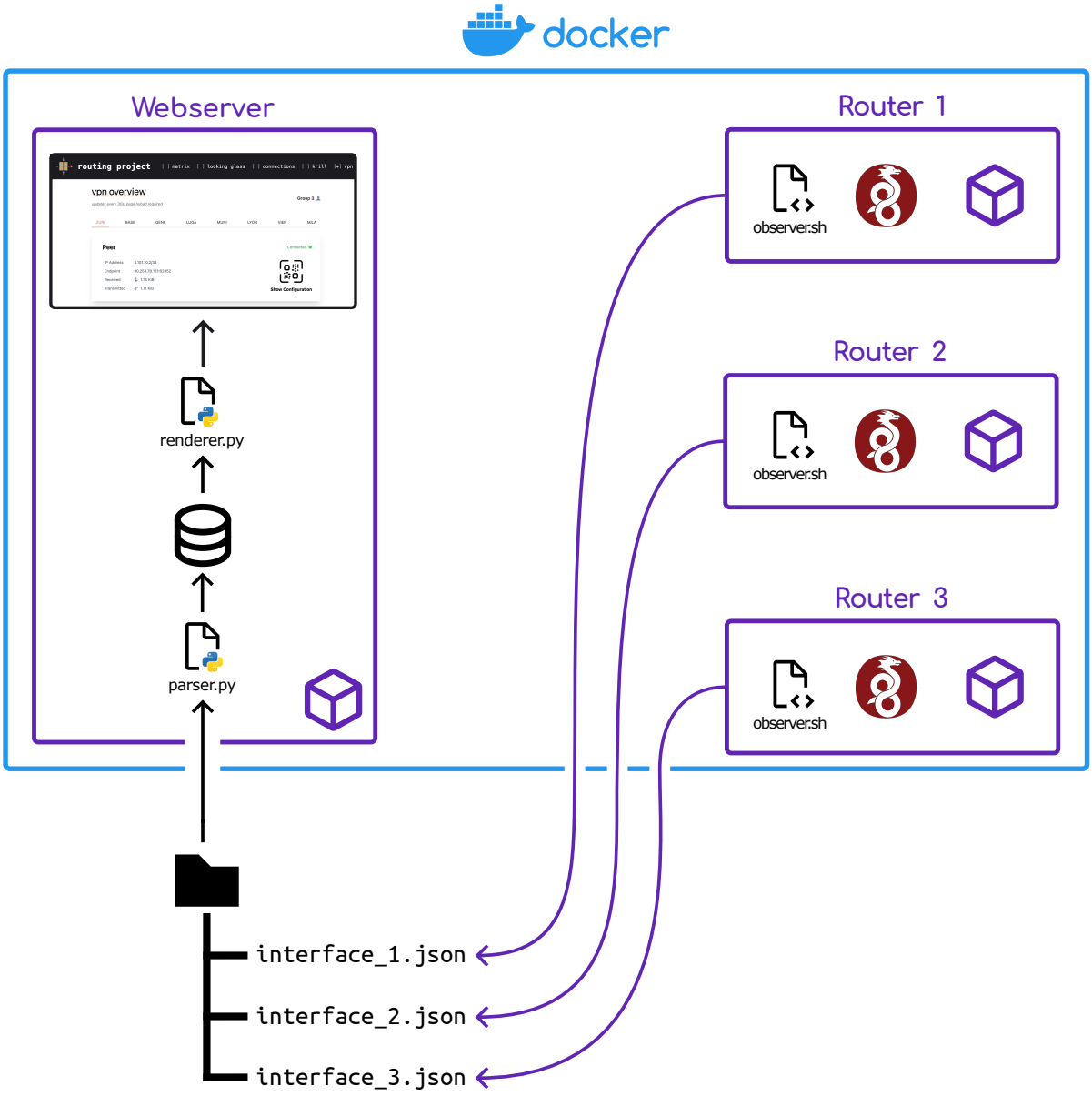


Figure 3.3: Schematic of the Observer System

Chapter 4

Evaluation

This chapter contains the performance analysis of the presented implementation.
The performance analysis consists of two parts. First, we compare the new version of the Mini-Internet (with VPN) against the old version (Section 4.1). Second, we send packets through the VPN and observe the CPU performance (Section 4.2).
Our analysis confirms that the VPN adds a minimal overhead. Limiting the traffic is necessary to ensure a stable CPU performance.

4.1 Passive Impact on CPU Performance

The passive impact tests aim to measure the computational overhead of the VPN. The test is "passive" in the sense that no traffic is sent over the VPN. Wireguard claims to be lightweight, as it is directly integrated into the kernel [5]. So as a result we expect very little computational overhead. The implementation also uses some CPU time for the observer processes (referring to Section 3.4). With these tests, we want to find a suitable configuration for the observers that does not cost too much computational time.

4.1.1 Setup

In the first test we set up two Mini-Internet instances - one with and one without VPN. We measure the CPU utilization using the System Activity Reporter (sar) from linux' sysstat package [11]. This tool will be used for all tests in this chapter. The CPU utilization is averaged over an interval of one minute. The test is repeated for Mini-Internet instances with different numbers of routers. Again, we average the CPU utilization over one minute.

In the second test, the number of routers in the Mini-Internet is fixed. We first measure the CPU utilization without VPN to get a baseline. Then, VPN is enabled with different observer sleep periods.

4.1.2 Results

The additions to the setup increase CPU utilization even if no traffic is sent over the VPN.
We can conclude that more routers lead to a higher CPU utilization (Figure 4.1). Wireguard adds additional load. However, it is relatively low (around 1% - 2%) compared to the total utilization. The overhead caused by wireguard seems to be fairly constant, even for higher number of routers (Figure 4.1).
A higher observer frequency results in higher CPU utilization (Figure 4.2). There is a tradeoff

between a responsive UI with frequent updates and CPU performance. A suitable option is a sleep period of 30s, which in this case results in an increase of 3 % in CPU utilization.

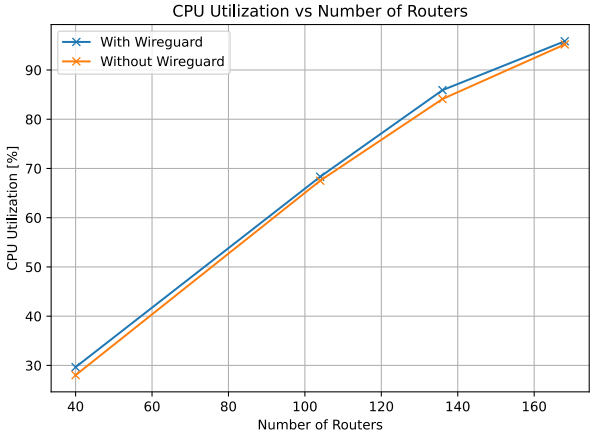


Figure 4.1: Impact of Network Size on CPU Utilization

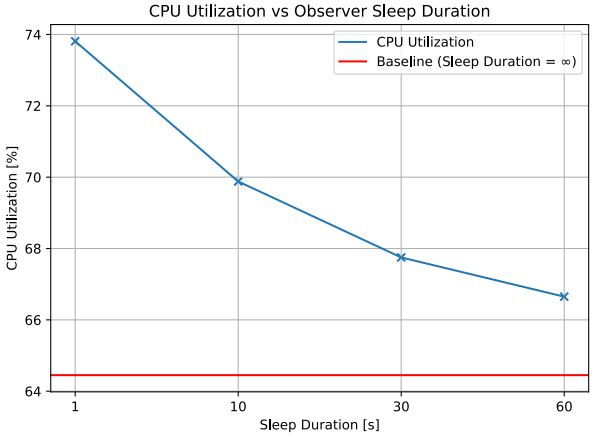


Figure 4.2: Impact of Wireguard Observer Period on CPU Utilization

4.2 Active Impact on CPU Performance

The Mini-Internets traffic consists of BGP/IGP advertisements and some student applications like traceroute and ping. With the VPN, it is expected to see an increase in traffic (for example due to students sending files or accessing services). This section aims to find out how the Mini-Internet can handle the additional traffic. In addition, we test different rate limit configurations. The goal is to find a configuration that ensures good CPU performance and therefore a good stability of the simulated network.

4.2.1 Setup

For the tests in this section, the Mini-Internet is set up with 20 ASes, resulting in a total of 104 routers. For generating the traffic we use iperf3 [12]. Iperf3 starts a client and a server session on two devices. The client sends random generated traffic over a TCP connection.
Figure 4.3 illustrates the iperf3 setup. We set up a second machine running docker. Then we create a pair of containers - a client and a server. Both containers connect to the Mini-Internet via Wireguard. They then sent traffic to each other over the VPN.
For the tests, we set up various number of iperf3 pairs. We then average the CPU utilization and the received bitrate (summed over all iperf3 sessions) over one minute.
For each test we set up limitations on the traffic volume. To find the best fit, we test the methods presented in Section 3.3 with different parameters.

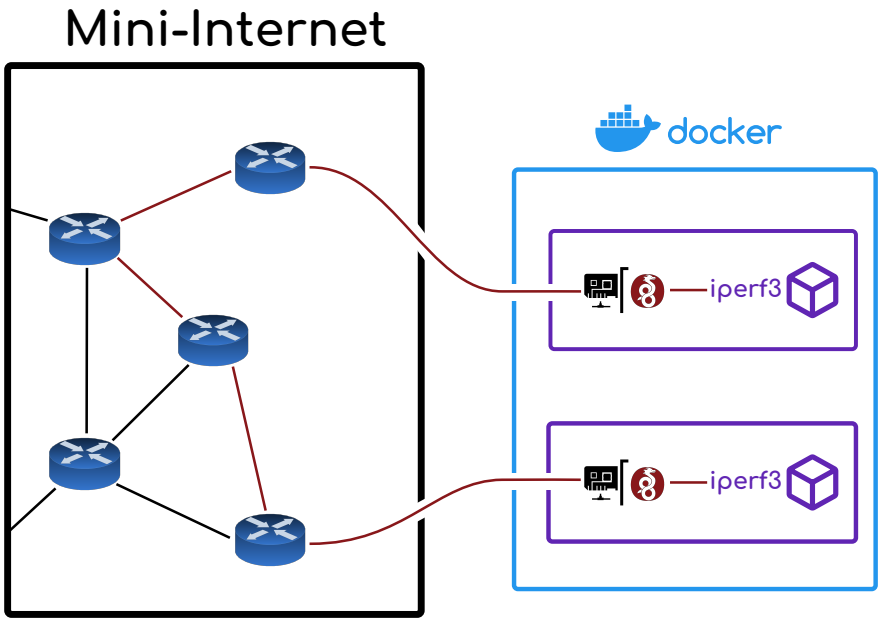


Figure 4.3: Iperf Test Setup

4.2.2 Results

Similar to [2], we notice that higher traffic volume leads to higher CPU utilization.
A plot of all bitrates shows a linear relation between the bitrate and CPU utilization (Figure 4.4). In order to limit the CPU utilization we therefore need to limit the traffic volume.

The tests show that 1 Gbit/s links lead to very high CPU usage (Figure 4.5). More restrictive limits on the links, as well as the Wireguard interfaces, yield better CPU utilization. Depending on the intended use case and the available computational power, the limits can be adjusted accordingly. As of now the students are not supposed to send a lot of traffic through the VPN. Therefore we can limit the Wireguard interfaces to 1 Mbit/s and the links to 10 Mbit/s.

Restricting the number of VPN connections is another option to ensure better CPU performance. In practice this should not be necessary. It is reasonable to assume that not many students simultaneously connect to the VPN. In the case of only a few connections, the bandwidth could be increased while maintaining a reasonable CPU load.

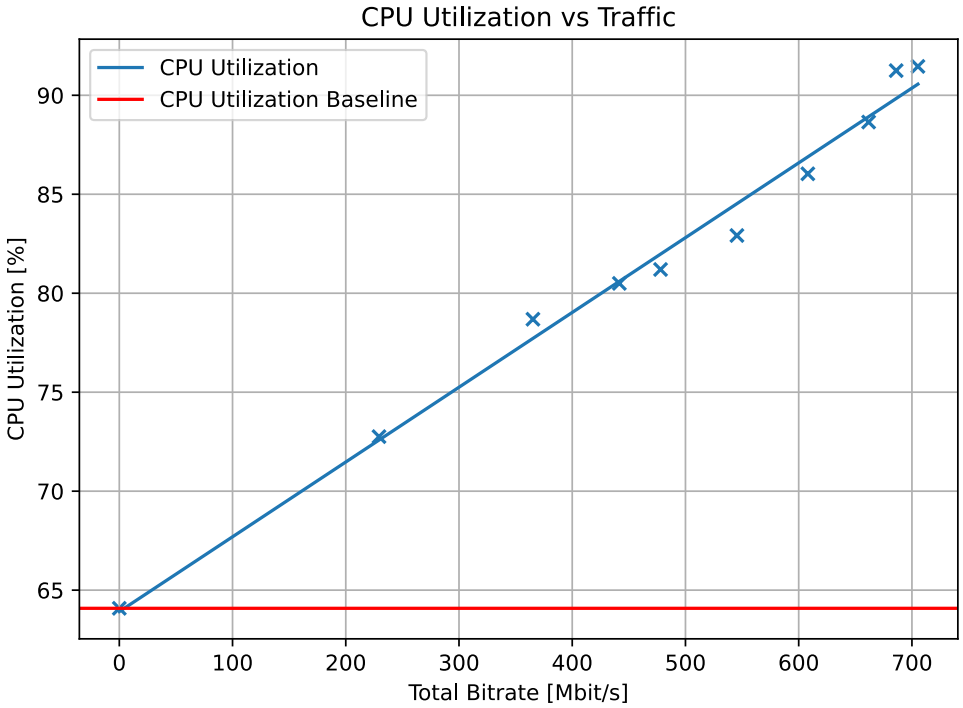


Figure 4.4: CPU Utilization vs Received Bitrate without restrictions

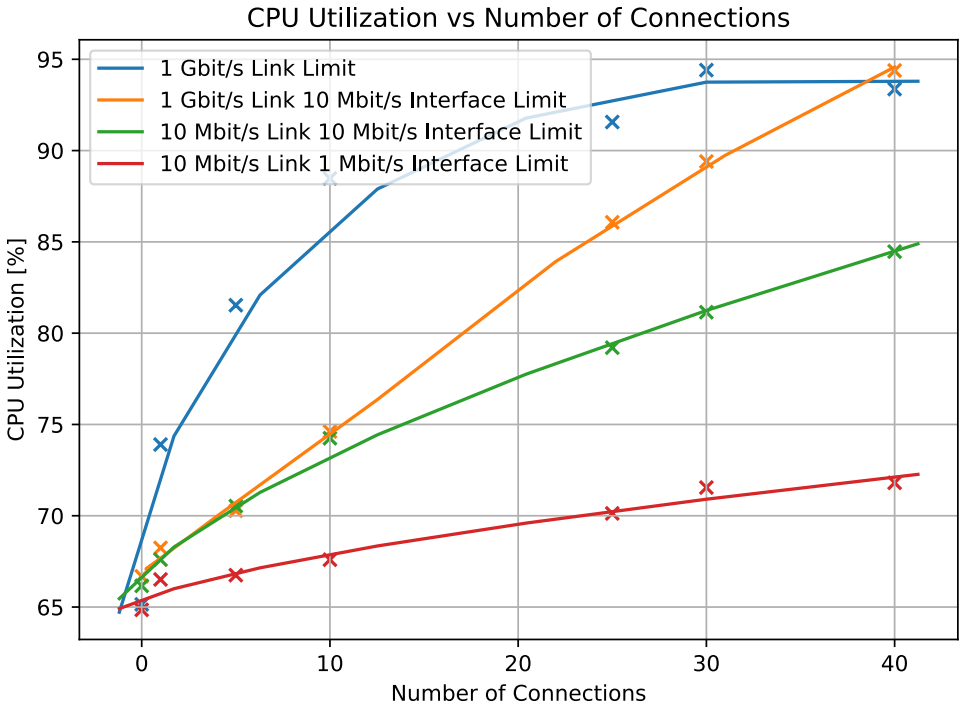


Figure 4.5: CPU Utilization vs Number of Connections with bandwidth and rate limits

Chapter 5

Outlook

The implementation was done and tested in the scope of this Bachelor thesis. In the weeks to come the project's code will be included into the official Mini-Internet codebase.
The goal was to use this work for the Communication Networks course (Section 1.1). So the next step is to integrate the VPN into the course and see how the students respond to it. Depending on their feedback and the teachers observations, adjustments and improvements can be made. This might include an updated version of the UI, or modifying the rate limits.

Further down the line, new VPN based services can be added to the course. One option is to let students host their own webservers. This could be done by setting up a simple webserver like Nginx or Apache. The webserver can be connected to the Mini-Internet vie Wireguard. Now, the server can be accessed from inside the Mini-Internet using its public IP.
Many other services could be included thanks to the new VPN. In the end, the lecturers of the Computer Networks course need to decide which services they want to include.

Two ideas were mentioned in Chapter 3. One is to dynamically add peers to existing Wireguard interfaces. The second one is to reduce the amount of exposed ports. This could be done by using routing tables to route packets to the right Wireguard interfaces.

Another idea was mentioned in the Mini-Internet paper [2]. ETH's Mini-Internet could be connected to other universities Mini-Internets. Now that the VPN is working, two Mini-Internets could be connected by establishing a Wireguard connection. Of course this would require additional considerations and adjustments. For example, it would be more realistic to not connect two routers via Wireguard, but use the IXPs instead.

It would also be interesting to explore use cases beyond the Mini-Internet. This work can be applied anywhere, where a private network of Docker containers needs to be accessed through a VPN. A typical scenario could be a private container network including a Network Attached Storage (NAS), a container for managing a smart home system, a CalDAV calendar and a media server. These services communicate via an internal Docker network, but have some insecure, non-encrypted interfaces. For security reasons, these interfaces should not be publicly available over the internet. The work presented in this thesis provides a solution to this problem. It allows to access the internal Docker network without exposing insecure interfaces.

Chapter 6

Summary

This report presents a method to access an isolated network of Docker containers. The Wireguard protocol is used to build a VPN that allows secure access to the Docker network. This solution was embedded into the Mini-Internet teaching platform.
Before this project, the Mini-Internet was isolated - only the routers could be accessed via SSH. Now, it is possible to connect other devices over the VPN. This feature allows students to actively use the Mini-Internet, which makes it feel more like using the "real" internet.
Connecting personal devices is straightforward. A user interface allows to establish a connection by scanning a QR code. The user interface is semalessly integrated into the existing webpage.

The necessary scripts only consist of around 100 lines of bash code. Additional tools allow for easy administration. To accomplish this task, a Wireguard interface is created outside of the container and then moved to the container's network namespace. Using this method, Wireguard can access the internet without Docker's port publishing methods. The Docker network remains isolated except for this interface.

A performance analysis showed that the feature has an overhead of around 2 % in CPU utilization. However, if excessive amounts of data are sent over the VPN, the analysis shows a major increase in CPU utilization. To tackle this problem, rate limits can be set up in the configuration file. They limit the bandwith to keep CPU utilization low and therefore insure network stability.

Moving forward, the VPN will be included into the Mini-Internet for the upcoming Communication Networks course. This practice test will provide feedback to modify and improve the VPN and user interface.

Bibliography

[1] ETH Networked Systems Group. NSG Website, 2025. https://nsg.ee.ethz.ch/.

[2] Thomas Holterbach, Tobias Bühler, Tino Rellstab, and Laurent Vanbever. An open platform to teach how the internet practically works. *SIGCOMM Comput. Commun. Rev.*, 2020.

[3] ETH Networked Systems Group. Mini-Internet GitHub Repository, 2025. https://github.com/nsg-ethz/mini_internet_project.

[4] Wikipedia. Virtual Private Network, 2025. https://en.wikipedia.org/wiki/Virtual_private_network.

[5] Jason A. Donenfeld. Wireguard: Next generation kernel network tunnel. In *Proceedings of the Network and Distributed System Security Symposium (NDSS)*. The Internet Society, 2017. https://www.wireguard.com/papers/wireguard.pdf.

[6] Jason A. Donenfeld. Wireguard Website, 2025. https://www.wireguard.com/.

[7] Docker. Bridge Network Driver, 2025. https://docs.docker.com/engine/network/drivers/bridge/.

[8] Flask Website, 2025. https://flask.palletsprojects.com/en/stable/.

[9] supervisord, 2025. https://supervisord.org/.

[10] Jason A. Donenfeld. Wireguard Network Namespaces, 2025. https://www.wireguard.com/netns/.

[11] Sysstat Linux Package, 2025. https://sysstat.github.io/.

[12] Iperf3 testsuite, 2025. https://iperf.fr/.

Appendix A

Implemetation Code

A.1 Wireguard Interface Initialization

```
#!/bin/bash

# Creates a new interface
create_if() {
  container_name=${1}
  listen_port=${2}
  ip_address=${3}

  # Path where the interface config file and keys are stored
  # This path must to be mounted to /etc/wireguard/ inside of the docker container
  path_to_file="${DIRECTORY}"/containers/${container_name}/wireguard

  # Variables can be imported from .env or another file
  VPN_LIMIT_ENABLED=false
  VPN_LIMIT_RATE="1mbit"
  VPN_LIMIT_BURST="32kbit"
  VPN_LIMIT_LATENCY="400ms"

  # Generate keys
  private_key=$(wg genkey)
  public_key=$(echo "${private_key}" | wg pubkey)

  # Save configuration and public key
  printf "[Interface]\nPrivateKey=%s\nListenPort=%s\n\n" ${private_key} ${
      listen_port} | tee "${path_to_file}"/interface.conf > /dev/null
  echo "${public_key}" | tee "${path_to_file}"/interface.pubkey > /dev/null

  PID=$(get_container_pid ${container_name})

  # Add wireguard interface
  ip link add vpn type wireguard

  # Move interface to the container
  ip link set vpn netns "${PID}"

  # Configure interface
  nsenter --net=/proc/"${PID}"/ns/net ip address add "${ip_address}" dev vpn
  docker exec -u root "${container_name}" wg setconf vpn /etc/wireguard/interface.
      conf
  nsenter --net=/proc/"${PID}"/ns/net ip link set vpn up
```

```
39
40    # Set up rate limits
41    if [[ ${VPN_LIMIT_ENABLED} == true  ]]; then
42      nsenter --net=/proc/"${PID}"/ns/net tc qdisc add dev vpn root tbf rate ${
            VPN_LIMIT_RATE} burst ${VPN_LIMIT_BURST} latency ${VPN_LIMIT_LATENCY}
43    fi
44
45    # Set firewall exception
46    ufw allow "${listen_port}" > /dev/null
47  }
```

Listing A.1: Wireguard Interface Setup

A.2 VPN Cleanup Process

```
1   #!/bin/bash
2
3   set -o errexit
4   set -o pipefail
5   set -o nounset
6
7   DIRECTORY="$1"
8
9   # Deletes an interface
10  delete_if() {
11    PID=${1}
12    PORT=${2}
13
14    # Delete interface
15    nsenter --net=/proc/"${PID}"/ns/net ip link del vpn
16
17    # This is the path were the configuration files are mounted (adjust for your own
          purposes)
18    path_to_file="${DIRECTORY}"/container/$PID/wireguard
19
20    # Delete coniguration files
21    rm -f "${path_to_file}"/*
22
23    # Remove firewall exception
24    ufw delete allow "${PORT}" > /dev/null
25  }
26
27  # Delete all interfaces
28  # The ${PIDS} list keeps track of all containers that have a wg interface.
29  # The ${WG_PORTS} list keeps track of the ports that the interface uses
30  # When implementing this, replace this with your own variables to keep track
31  delete_all_ifs() {
32    for ((k = 0; k < group_numbers; k++)); do
33      PID=(${PIDS[$k]})
34      PORT=(${WG_PORTS[$k]})
35      echo "Deleting interface in container ${PID}"
36      delete_if "${PID}"  "${PORT}"
37    done
38  }
39
40  # Call function
41  delete_all_ifs
```

```
42
43 # Delete the database used for the UI
44 rm -f "${DIRECTORY}"/"${VPN_DB_FILE}"
45
46 # Delete the users passwords for the UI
47 rm -f "${DIRECTORY}"/"${VPN_PASSWD_FILE}"
```

Listing A.2: VPN Cleanup Process

A.3 Observer Process Code

```
1  #!/bin/bash
2
3  print_json(){
4    # The following snippet was taken from https://github.com/WireGuard/wireguard-
       tools/blob/master/contrib/json/wg-json and modified using chatgpt
5    # SPDX-License-Identifier: GPL-2.0
6    # Copyright (C) 2015-2020 Jason A. Donenfeld <Jason@zx2c4.com>. All Rights
       Reserved.
7
8    # Reset all state-tracking variables
9    last_device=""
10   first_device=true
11   first_peer=true
12   old_ifs=""
13
14   # Parses the wg dump into a json format
15   printf '{\n'
16   first_device=true
17   while read -r -d $'\t' device; do
18   if [[ $device != "$last_device" ]]; then
19       if ! $first_device; then
20     printf '\n\t\t}\n\t},\n'
21       fi
22       last_device="$device"
23       read -r private_key public_key listen_port fwmark
24       printf '\t"%s": {\n' "$device"
25       [[ $private_key != "(none)" ]] && printf '\t\t"privateKey": "%s",\n' "
           $private_key"
26       [[ $public_key != "(none)" ]] && printf '\t\t"publicKey": "%s",\n' "
           $public_key"
27       [[ $listen_port != "0" ]] && printf '\t\t"listenPort": %u,\n' "$listen_port"
28       printf '\t\t"peers": {\n'
29       first_device=false
30       first_peer=true
31   else
32       read -r public_key preshared_key endpoint allowed_ips latest_handshake
           transfer_rx transfer_tx persistent_keepalive
33       if ! $first_peer; then
34     printf ',\n'
35       fi
36       printf '\t\t\t"%s": {\n' "$public_key"
37       [[ $preshared_key != "(none)" ]] && printf '\t\t\t\t"presharedKey": "%s",\n'
           "$preshared_key"
38       [[ $endpoint != "(none)" ]] && printf '\t\t\t\t"endpoint": "%s",\n' "
           $endpoint"
```

```
      [[ $latest_handshake != "0" ]] && printf '\t\t\t\t"latestHandshake": %u,\n' "$latest_handshake"
      [[ $transfer_rx != "0" ]] && printf '\t\t\t\t"transferRx": %u,\n' "$transfer_rx"
      [[ $transfer_tx != "0" ]] && printf '\t\t\t\t"transferTx": %u,\n' "$transfer_tx"
      [[ $persistent_keepalive != "off" ]] && printf '\t\t\t\t"persistentKeepalive": %u,\n' "$persistent_keepalive"
      printf '\t\t\t\t"allowedIps": [\n'
      if [[ $allowed_ips != "(none)" ]]; then
    old_ifs="$IFS"
    IFS=,
    first_ip=true
    for ip in $allowed_ips; do
        if ! $first_ip; then
      printf ',\n'
        fi
        printf '\t\t\t\t\t"%s"' "$ip"
        first_ip=false
    done
    IFS="$old_ifs"
      fi
      printf '\n\t\t\t\t]\n\t\t\t}'
      first_peer=false
  fi
    done
    printf '\n\t\t}\n\t}\n}\n'
}
# Observer Process loop
while true
do
  rm -f /etc/wireguard/status.json
  # Print dump to temporary file first
  wg show all dump > /tmp/wg_dump.txt
  # Then, parse it
  if [[ -s /tmp/wg_dump.txt ]]; then
    print_json < /tmp/wg_dump.txt > /etc/wireguard/status.json
  fi
  sleep ${VPN_OBSERVER_SLEEP}
done
```

Listing A.3: Observer Process

A.4 Wireguard Configuration File

```
[Interface]
PrivateKey=wLz9IczIxad8OxBd3LWZKj9bP/KRlhsyZDuvo9EdUGk=
Address=3.106.10.2/32
DNS=198.3.0.2

[Peer]
PublicKey=h7YodRPy+Ug7pNs8v4xnJZyH/HkmQ6BhVMXl3500KDw=
AllowedIPs=0.0.0.0/0
Endpoint=ee-tik-nsgvm073.ethz.ch:15003
PersistentKeepalive=25
```

Listing A.4: Example for a Wireguard Client Configuration

Appendix B

Performance Analysis Test Setup

Hardware:

- CPU: AMD Epyc-Rome, 16 cores, 2.25 GHz
- RAM: 32 GB
- OS: Ubuntu Server 22.04.5 LTS (KVM)
- NIC: Emulated (virtio)

Software:

- CPU utilization meassured using the System Activity Reporter (sar) from linux' sysstat package
- Network performance meassured using iperf3