Starlink Analysis

July 24, 2021

This document was created by the research group ROADMAP-5G at the Carinthia University of Applied Sciences (CUAS) in June 2021. The used Starlink dish and modem were kindly provided by Stereo Media.

Christoph Uran (c.uran@fh-kaernten.at), Kurt Horvath (k.horvath@fh-kaernten.at), Helmut Wöllik (h.woellik@fh-kaernten.at)

1 Introduction

Since the emergence of 4G and 5G in the last years it seemed that for consumers, Internet access was a settled field. SpaceX with its division of Starlink did introduce a new satellite-based solution. Starlink is a system enabling satellite-based Internet access in selected areas of the Earth. The system is provided by SpaceX and consists of three parts:

1. The satellites in Low Earth Orbit (LEO). Currently, 1,635 of 11,914 planned satellites are deployed and operational. They are located at an altitude between 540 and 570 km (Source).
2. The ground stations are in constant communication with the satellites. They are necessary for providing Internet access and control information. There are currently 92 operational ground stations (Source). They are connected to so-called PoPs (Points of Presence), which act as interconnection points to the public data network (referred to as *Internet* in the rest of the document). There are currently 52 operational PoPs (Source)
3. The user dishes and routers for providing communication between the user devices and the satellites. The router is equipped with a Gigabit Ethernet port and WiFi for providing connectivity to the user devices. The dish is connected to the router and is powered using Power over Ethernet (PoE). The dish aligns itself towards the north to achieve the best possible reception to potentially visible satellites.

The following figure shows an example of the initial measurement setup.

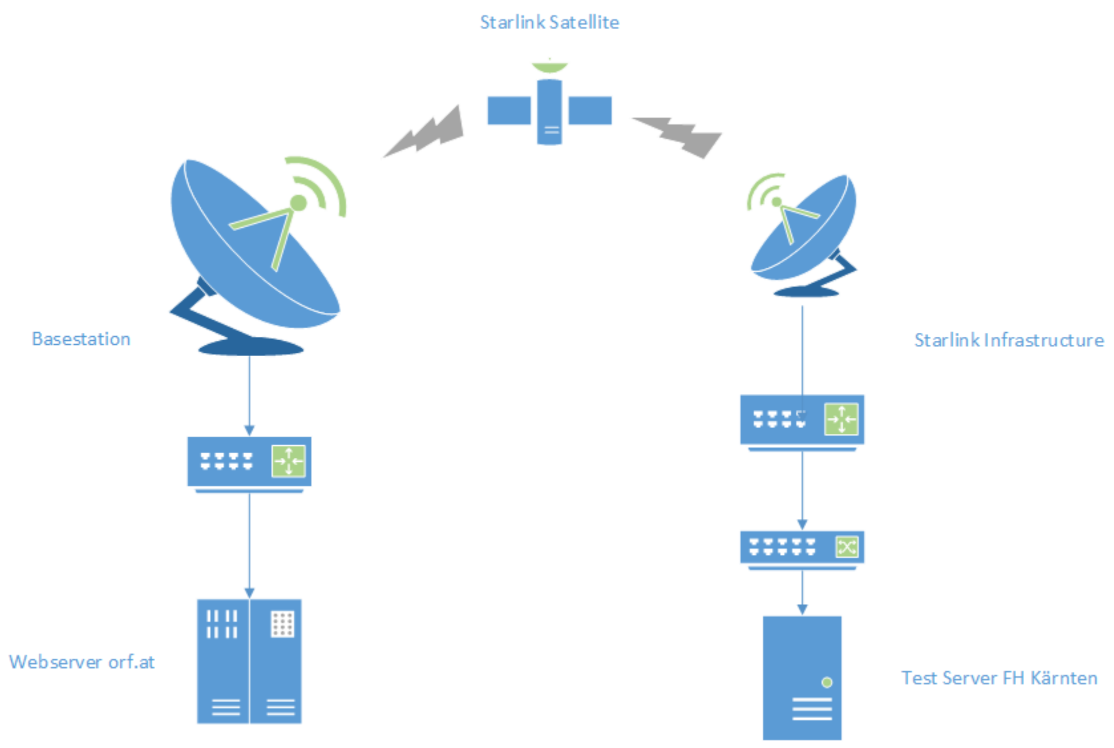


Figure 1: Initial measurement setup and description of a Starlink-based Internet connection

The whole setup, as it has been granted by the United States Federal Communications Commission (FCC) can be found here.

In summary, the most important findings of our experiments are:

- An average download throughput of approximately 170 Mbit/s and a maximum download throughput of approximately 330 Mbit/s could be reached in a time window of about 7 hours.
- An average upload throughput of approximately 17 Mbit/s and a maximum upload throughput of approximately 60 Mbit/s could be reached in a time window of about 15 hours.
- The observed latencies to a server in Vienna vary widely between less than 30 milliseconds and over 2 seconds.
- In approximately 98% of the time, the latency is below 90 milliseconds and in approximately 77% of the time, the latency is below 50 milliseconds.
- During a continuous ping test (one ping per second) for nearly 7 days, a downtime of 2.4% could be observed. This percentage is expected to decrease when reducing the ping intervals to less than 1 second.
- Steady continuous video streaming via YouTube delivers a satisfactory experience. In very rare cases there might be some short interruptions of up to 4-6 seconds.
- The automatic switching between the satellites seems to follow a pre-defined timing of 15 seconds. This means that changes in latencies (for better or worse) nearly always occurs between these 15-second windows.
- No conclusive evidence for a correlation between the current satellite constellations and the observed outages could be found. This does not mean that this correlation doesn't exist, it rather means that this topic requires further investigation.
- In an observed period of approximately 56 hours, the power consumption of the whole client-

side system (router and satellite dish) was 105 Watts on average with a maximum of 190 Watts.
- Officially, Starlink Internet access is only offered in selected regions. However, we could prove that the connection also works outside these regions at four tested positions.
- The time between powering the system up and having a working Internet connection varies between 5 and 20 minutes. The factors that influence that time have not been evaluated so far.
- As far as we could observe, the public IP address assigned to a router remains the same. In our case, the address was 188.95.144.107.

The rest of this document describes the data that has been acquired using the provided Starlink system and interprets the interesting findings. For reproducibility, the document also includes the source code used for the evaluation.

2 Measurement Setup

For different measurements, the measurement setups looked slightly different. In every case, the Starlink satellite dish was positioned with a clear view of the northern sky, which is a requirement according to the installation instructions. Depending on the measurement there were different computers (Windows and Linux computers, servers in the cloud, and mobile phones connected to Starlink's WiFi), different measurement programs (`ping`, `iperf`, and simple connectivity test tools like Speedtest or RTR Netztest), and different logging methods involved. The detailed measurement setup for each type of measurement is described below. The setup at the building of CUAS (campus Klagenfurt Primoschgasse) is shown below. Unless stated otherwise, all the measurements have been conducted there.

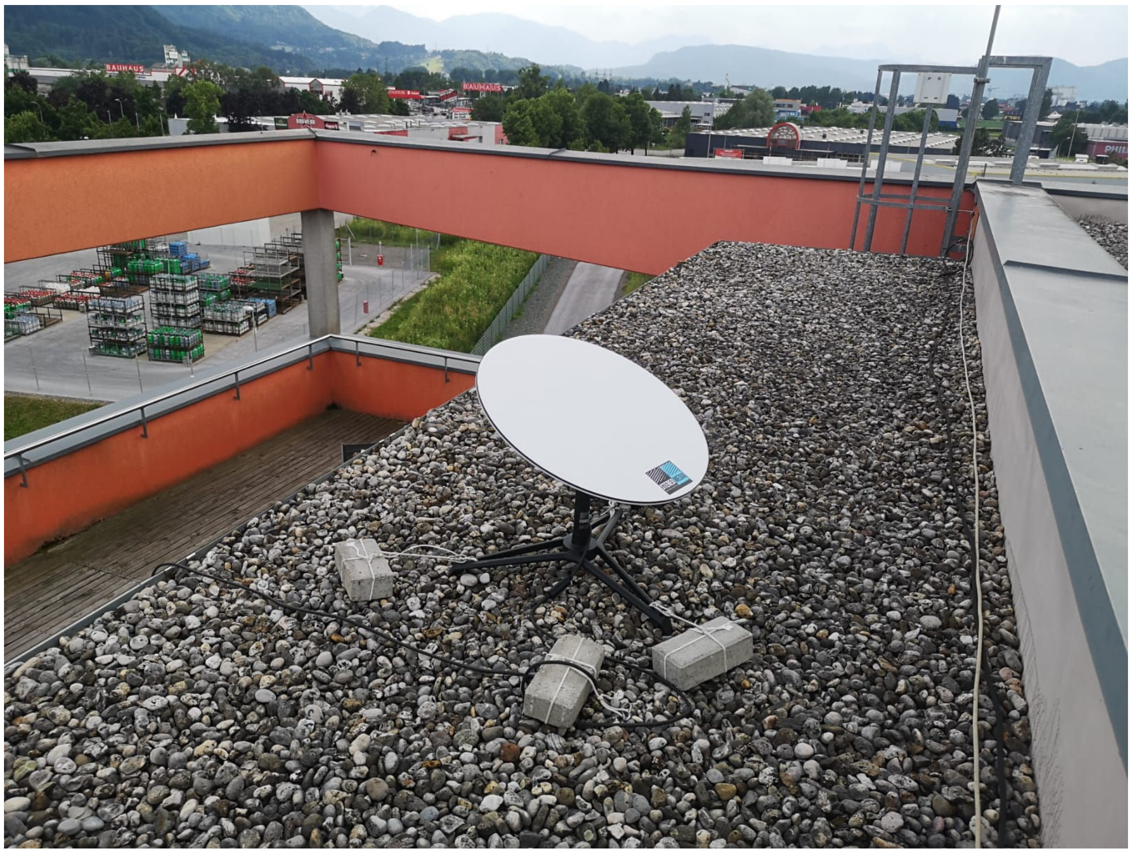


Figure 2: Satellite dish setup on the roof of the CUAS building in Klagenfurt

3 Throughput

The throughput is one of the most important metrics of an Internet connection. It gives information about the achievable speed of the connection from the server to the client (*download*) as well as from the client to the server (*upload*). In most cases, these two values vary quite a lot, because most Internet connections are not symmetric, but asymmetric. The values are given in *bits* (or a multiple of that, e.g. *K* for kilo, *M* for mega, ...) per second. In this document, we always refer throughput to as megabits per second or *Mbit/s*.

For throughput measurements, the program `iperf` has been used. By default, this program uses TCP for the data transmission, which also applies to the measurements in this document. As previous measurements in other projects have shown, it is important to use the same version of `iperf` on both sides. In our case, version 3.9 was used on both sides. Also, both sides have Ubuntu 20.04 LTS installed. The client-side consisted of a Lenovo x230 notebook, which was connected to the Starlink router via a Gigabit-capable switch by TP-Link. The server side consisted of a server hosted in the Frankfurt datacenter of the Google Cloud. Frankfurt was chosen because the most likely (or only possible) PoP when connecting to the Internet via Starlink in Austria is also located in Frankfurt. The measurement procedure was to continuously measure the up- or download throughput until the measurement program was stopped. This resulted in log files (formatted in JSON) with measurement results available for each second. The measurement results were logged for up- and download on the client- and server-side (effectively resulting in four log files). However, only the client-side was relevant for the download measurement and only the server-side was relevant for the upload measurement. Furthermore, the observed data rates should be the same on the client and the server, when using TCP.

3.1 Importing Libraries and reading Data

The following Python code block

- imports the necessary libraries,
- sets some initial parameters for plotting graphs,
- loads the log files mentioned above,
- parses the log file into more appropriate data structures using an external `helper` program and
- prints some preliminary information about the measured data.

[1]:
```
# imports
import json
import matplotlib.pyplot as plt
import seaborn as sns
import helpers
import statistics

# configuration of the size of the plotted figures
plt.rcParams['figure.figsize'] = [12, 6]

# read download performance statistics as seen from the client side
file = open('starlink_iperf_tests/iperf-client-download-20210614.log')
download_clientside = json.load(file)
file.close()

# read upload performance statistics as seen from the server side
file = open('starlink_iperf_tests/
 ↪iperf3-server-test-from-client-to-server-20210614.log')
upload_serverside = json.load(file)
file.close()

# extract necessary fields from data
dl_timestamps_clientside, dl_throughputs_clientside, dl_retransmits_clientside,␣
 ↪dl_cwnd_clientside = helpers.extract_ts_and_throughput(download_clientside,␣
 ↪unit='M')
ul_timestamps_serverside, ul_throughputs_serverside, ul_retransmits_serverside,␣
 ↪ul_cwnd_serverside = helpers.extract_ts_and_throughput(upload_serverside,␣
 ↪unit='M')

print('Download Throughput: Average %.1f Mbit/s, Maximum %.1f Mbit/s, Mean %.1f␣
 ↪Mbit/s' % (statistics.mean(dl_throughputs_clientside),␣
 ↪max(dl_throughputs_clientside), statistics.
 ↪median(dl_throughputs_clientside)))
```

```
print('Upload Throughput: Average %.1f Mbit/s, Maximum %.1f Mbit/s, Mean %.1f␣
↪Mbit/s' % (statistics.mean(ul_throughputs_serverside),␣
↪max(ul_throughputs_serverside), statistics.
↪median(ul_throughputs_serverside)))
```

```
Download Throughput: Average 169.6 Mbit/s, Maximum 332.4 Mbit/s, Mean 177.0
Mbit/s
Upload Throughput: Average 17.1 Mbit/s, Maximum 59.4 Mbit/s, Mean 17.8 Mbit/s
```

Alongside other information, the output shows that the achievable download throughput is about 170 Mbit/s on average in download and about 17 Mbit/s on average in upload.

3.2 Download Test Results

For a more detailed look at the download throughput, a box plot has been generated, showing the statistical variance of the measured data rates for each hour of the measurement time frame. A box plot is a way to show the statistical distribution of data, with which it is easy to see the minimum values, maximum values, medians, and quartiles. The outliers have been ignored for the following box plots because they provide no statistically relevant information and would make the plots more difficult to read and interpret.

[2]:
```
# defining start and end time, getting the respective indices and processing␣
↪the data
start_time = '2021-06-14 10:00:00'
end_time = '2021-06-14 18:00:00'
start_index, end_index = helpers.get_indices_by_time(dl_timestamps_clientside,␣
↪start_time, end_time)
boxes, times, unsuccessfuls = helpers.
↪generate_latency_boxes(dl_timestamps_clientside, dl_throughputs_clientside,␣
↪start_index, end_index)

# creating x-axis labels
labels = []
for time_and_unsuccessfuls in zip(times, unsuccessfuls):
    labels.append('%s\n(%s)' % time_and_unsuccessfuls)

# plotting the data
sns.set(context='notebook', style='whitegrid')
ax = sns.boxplot(data=boxes, showfliers=False)
ax.set(xlabel='Hour\n(seconds without connectivity)', ylabel='Throughput (MBit/
↪s)')
ax.grid(b=True, which='minor', linewidth=0.5)
plt.xticks(plt.xticks()[0], labels)
plt.ylim(0,)
_p = plt.title('Download Throughput between %s and %s)' %␣
↪(dl_timestamps_clientside[start_index], dl_timestamps_clientside[end_index]))
```

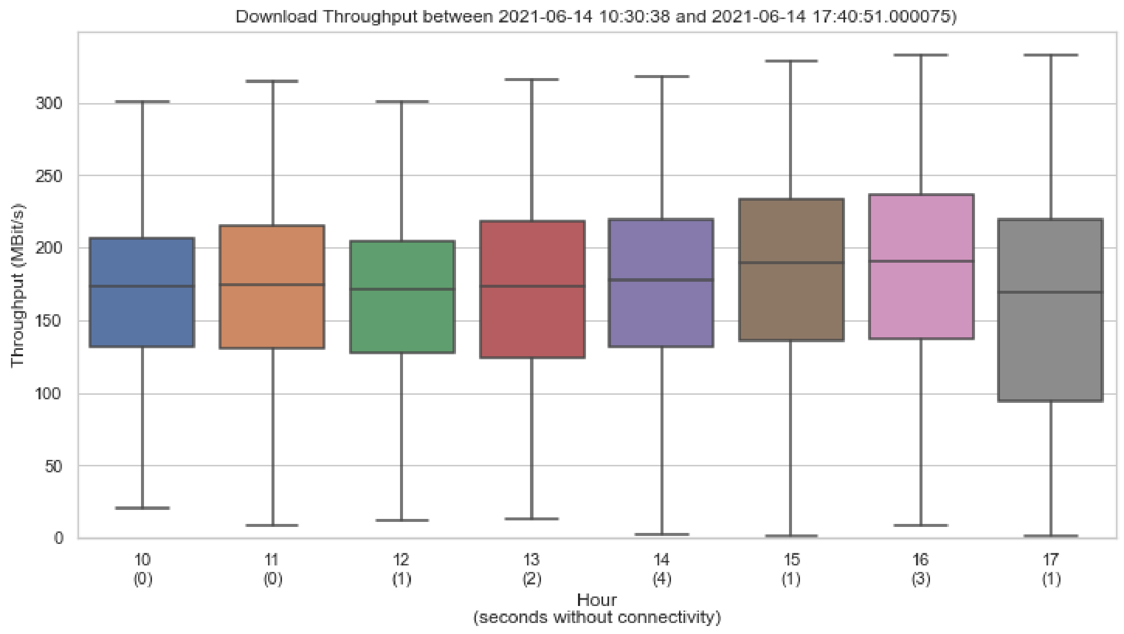


Figure 3: Download throughput in the period of approximately 7 hours

The measurement shows that the download throughput varies widely between 0 and 330 Mbit/s, whereas the median throughput has remained relatively constant at approximately 175 Mbit/s across the whole measurement period. In addition, there are hardly any seconds without connectivity.

3.3 Upload Test Results

The same has been done with the data generated at the upload throughput test.

```
[3]: # defining start and end time, getting the respective indices and processing␣
      ↪the data
     start_time = '2021-06-14 17:00:00'
     end_time = '2021-06-15 09:00:00'
     start_index, end_index = helpers.get_indices_by_time(ul_timestamps_serverside,␣
      ↪start_time, end_time)
     boxes, times, unsuccessfuls = helpers.
      ↪generate_latency_boxes(ul_timestamps_serverside, ul_throughputs_serverside,␣
      ↪start_index, end_index)

     # creating x-axis labels
     labels = []
     for time_and_unsuccessfuls in zip(times, unsuccessfuls):
         labels.append('%s\n(%s)' % time_and_unsuccessfuls)

     # plotting the data
```

```
sns.set(context='notebook', style='whitegrid')
ax = sns.boxplot(data=boxes, showfliers=False)
ax.set(xlabel='Hour\n(seconds without connectivity)', ylabel='Throughput (MBit/
 ↪s)')
ax.grid(b=True, which='minor', linewidth=0.5)
plt.xticks(plt.xticks()[0], labels)
plt.ylim(0,)
_p = plt.title('Upload Throughput between %s and %s)' %␣
 ↪(ul_timestamps_serverside[start_index], ul_timestamps_serverside[end_index]))
```

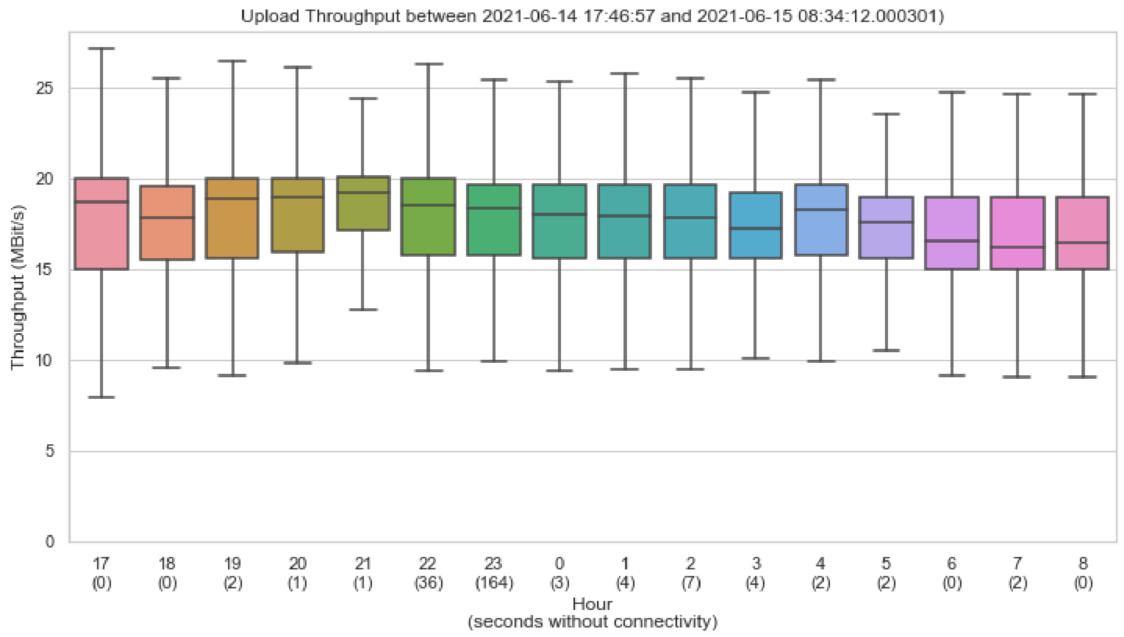


Figure 4: Upload throughput in the period of approximately 15 hours

Also here, the upload throughput varies widely between roughly 10 and 25 Mbit/s, whereas the median throughput remains relatively constant. It is notable, that the throughput hardly ever drops below 10 Mbit/s, which makes the system suitable for upload video streaming in some use cases. It is noteworthy, however, that there were unusually many seconds without connectivity between 22:00 and 00:00. As - unfortunately - the logging of the satellite constellation and the logging of the latencies/outages has not been active at this point, this behavior can not be investigated any further for now.

4 Latencies

Besides the data rate, latency is an important metric for a telecommunication network. In this report, the term latency is understood as the Round Trip Time (RTT), which is the two-way latency. In the following analysis, the latency has been measured using the `ping` command on the Lenovo x230 notebook stated above. In its default configuration `ping` issues an ICMP echo request to a given server and waits for an appropriate reply. It does so every second and also measures the time it takes for the reply to arrive. This time is the *latency*. If the reply arrives, the server is considered to be *reachable*, if not, it is *unreachable*. In our setup, the `ping` command was configured in a way to also include the exact timestamp of the reply and to log the output of the command to a file. This log file is analyzed in the following code blocks.

4.1 Importing Libraries and reading Data

The following code block

- imports the necessary libraries,
- sets some initial parameters for plotting graphs,
- initializes variables,
- loads the log file mentioned above and
- parses the log file into more appropriate data structures.

[4]:
```
# imports
from datetime import datetime, timedelta
import time
import matplotlib.pyplot as plt
import seaborn as sns
import re
import helpers

# configuration of the size of the plotted figures
plt.rcParams['figure.figsize'] = [12, 6]

# declaring and initializing variables
all_timestamps = []
all_status = []
all_latencies = []
unsuccessful_ts = []
colors = []
working_windows = []
not_working_windows = []
latency_after_not_working = []
is_working = None
state_since = None

# opening and reading the log file
log_file = open('starlinktest_with_load_20210602_to_20210609.log', 'r')
log_lines = log_file.readlines()

for l in log_lines:
    try:
        # get the time of the current line
        ts = datetime.fromtimestamp(float(re.search('\[(.*)\]', l).group(1)))
        # ignore certain lines
        if 'Destination Host Unreachable' in l or 'Destination Net Unreachable'␣
↪in l:
            pass
        elif 'no answer yet for' in l:
            all_timestamps.append(ts)
            all_status.append(1)
            unsuccessful_ts.append(ts)
            all_latencies.append(0)
            colors.append('r')
            if is_working == None:
                is_working = False
                state_since = ts
            elif is_working == True:
                is_working = False
                working_windows.append(ts - state_since)
                state_since = ts
        else:
            all_timestamps.append(ts)
            all_status.append(0)
            all_latencies.append(float(re.search(' time=(.*) ms', l).group(1)))
            colors.append('b')
            if is_working == None:
                is_working = True
                state_since = ts
            elif is_working == False:
                is_working = True
                not_working_windows.append(ts - state_since)
                state_since = ts
                latency_after_not_working.append(float(re.search(' time=(.*)␣
↪ms', l).group(1)))
    # if an error happens, ignore the error but write a log message
    except (ValueError, AttributeError):
        print('Skipping line: %s' % l)
```

```
Skipping line: PING 194.232.104.3 (194.232.104.3) 56(84) bytes of data.
```

As can be seen here, only the first line (`PING 194.232.104.3 ...`) has been skipped and all the other lines have been processed correctly. In this test, it has been decided to ping one of the servers of ORF.

4.2 Long-term Latency Measurements

The following plot shows the latencies (blue dots) and outages (red dots) across the whole observation period.

[5]:
```
plt.scatter(all_timestamps, all_latencies, c=colors, alpha=0.1)
plt.title('Long-term Starlink Latency Measurements')
plt.xlabel('Time')
plt.ylabel('Latency (ms)')
plt.grid(True)
plt.show()
```

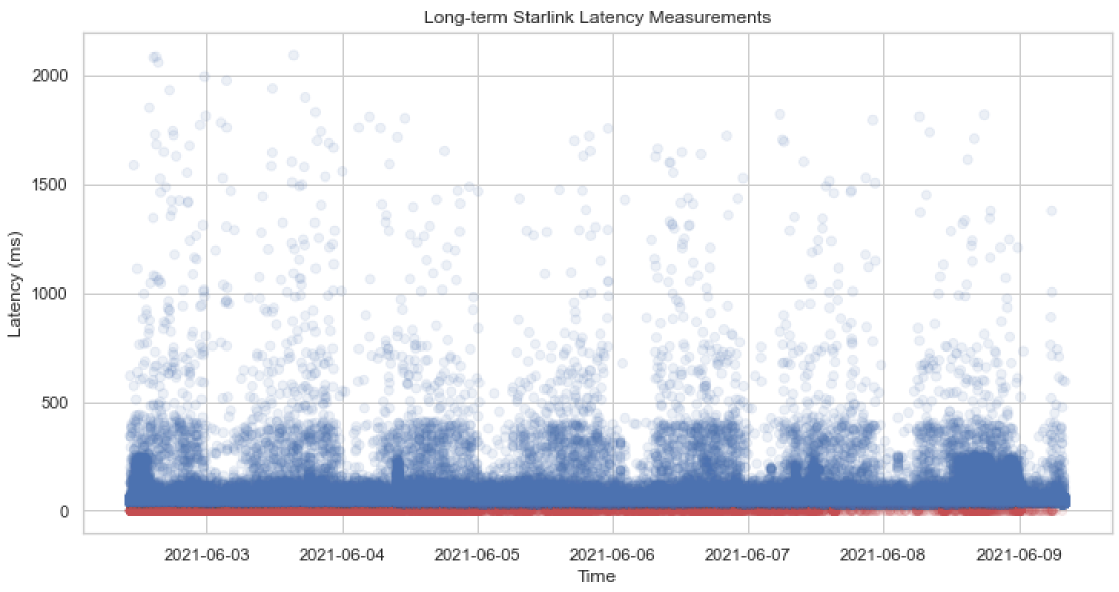


Figure 5: Long-term Starlink latency measurements in a period of approximately one week

The first interesting observation is that there are fewer extremely high latencies (more than 200 ms) in the nights.

4.3 Statistical Analysis of the Measured Data

The following plot observes the fluctuation in latencies by night in more detail. For a better analysis, the presentation of the data is done using a box plot (one box per hour). This makes it easier to interpret the results.

[6]:
```
# defining start and end time, getting the respective indices and processing␣
↪the data
start_time = '2021-06-04 12:00:00'
end_time = '2021-06-05 12:00:00'
start_index, end_index = helpers.get_indices_by_time(all_timestamps,␣
↪start_time, end_time)
boxes, times, unsuccessfuls = helpers.generate_latency_boxes(all_timestamps,␣
↪all_latencies, start_index, end_index)

# creating x-axis labels
labels = []
for time_and_unsuccessfuls in zip(times, unsuccessfuls):
    labels.append('%s\n(%s)' % time_and_unsuccessfuls)

# plotting the data
sns.set(context='notebook', style='whitegrid')
ax = sns.boxplot(data=boxes, showfliers=False)
ax.set(xlabel='Hour\n(number of lost pings)', ylabel='Latency (ms)')
ax.grid(b=True, which='minor', linewidth=0.5)
plt.xticks(plt.xticks()[0], labels)
_p = plt.title('Analysis of the Latency between %s and %s' %␣
 ↪(all_timestamps[start_index], all_timestamps[end_index]))
```

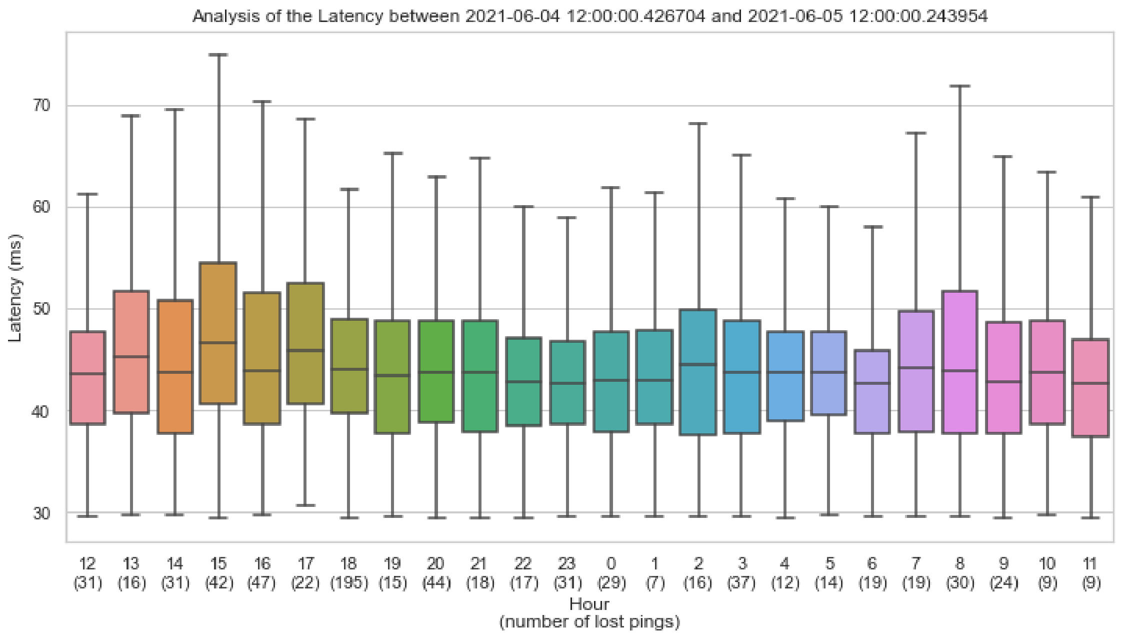


Figure 6: Analysis of the latency in a period of 24 hours

The plot shows that the vast majority of latencies are the same by night and by day and remains below approximately 70 ms. Therefore, the effect seen above can be considered to be unproblematic.

4.4 Cumulative Latency Distribution

To get a better understanding of which latencies to expect from a Starlink-based Internet connection, the following plot shows a cumulative histogram of the latencies measured above.

[7]:
```
lat_cap = 150
all_working_latencies = [l if l < lat_cap else lat_cap for l in all_latencies␣
 ↪if l != 0]
plt.hist(all_working_latencies, bins=lat_cap*10, density=True, histtype='step',␣
 ↪cumulative=True)
plt.title('Cumulative Histogram of the Latencies capped at %s ms' % lat_cap)
plt.xlabel('Latency (ms)')
plt.ylabel('Frequency')
plt.xlim([40, 149])
plt.ylim([0.5, 1.03])
plt.grid(True)
plt.minorticks_on()
plt.grid(b=True, which='minor', color='#999999', linestyle='-', alpha=0.2)
plt.show()
```

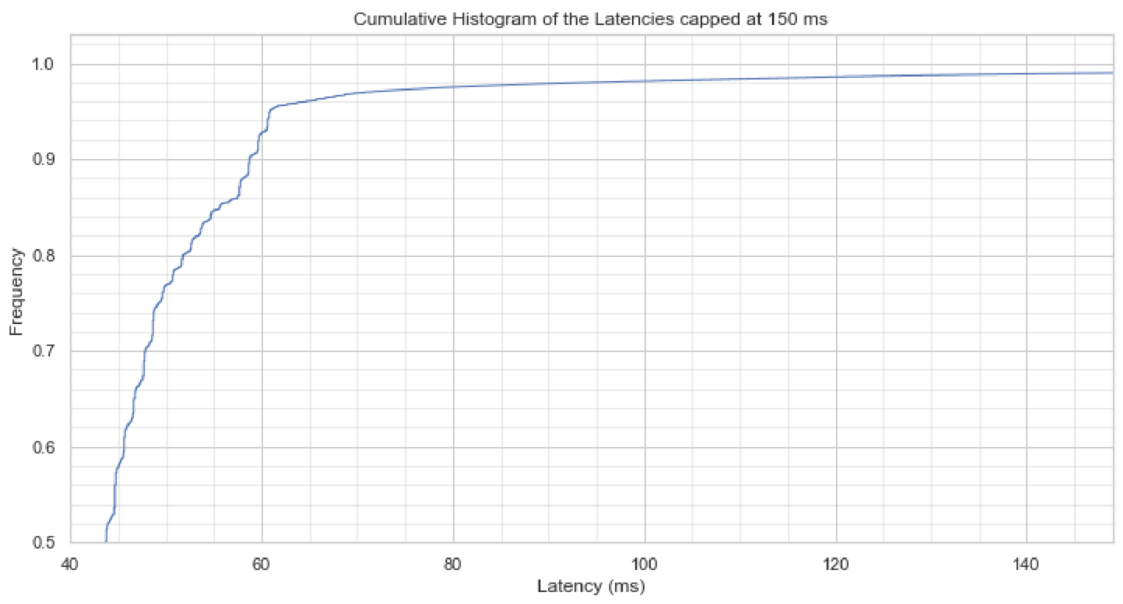


Figure 7: Cumulative histogram of the recorded latencies

The following example shows, how this plot can be most efficiently read: If a certain use case requires a maximum latency of 60 ms, the plot shows that this can be achieved by 93% of the observed packets. Then the decision can be made if this is enough to satisfy the requirements.

5 Outages

Having a network outage is something that should be avoided as much as possible. Therefore, it is also necessary to know the probability that an outage occurs using a Starlink-based Internet connection and also how such an outage manifests itself. The following analysis uses the same data that has been used for the latency evaluation before.

5.1 Some general Information on the Measured Data

The following code block prints a descriptive analysis of the gathered data and displays the respective down- and uptimes in the observed time frame.

```
[8]: # crunching the numbers
all_pings = len(all_timestamps)
failed_pings = len(unsuccessful_ts)
time_working = sum(working_windows, timedelta())
time_not_working = sum(not_working_windows, timedelta())

# printing the statistics
print('It has been pinged from %s to %s UTC' % (all_timestamps[0],␣
 ↪all_timestamps[-1]))
print('From a total of %d pings, %d have been lost' % (all_pings, failed_pings))
print('This is a loss of %.2f%%' % ((failed_pings / all_pings) * 100))
print('The amount of time there was connectivity was %s' % time_working)
print('The amount of time there was no connectivity was %s' % time_not_working)
print('This is a no-connectivity-percentage of %.2f%%' % ((time_not_working /␣
 ↪(time_working + time_not_working)) * 100))
```

```
It has been pinged from 2021-06-02 10:26:35.126786 to 2021-06-09 08:07:41.022011
UTC
From a total of 595484 pings, 17784 have been lost
This is a loss of 2.99%
The amount of time there was connectivity was 6 days, 17:20:01.622445
The amount of time there was no connectivity was 3:54:51.136815
This is a no-connectivity-percentage of 2.37%
```

In some preliminary tests with shorter ping intervals (0.1 seconds) it could be observed, that the amount of time there was no connectivity (and therefore also the no-connectivity-percentage) decreased in comparison to the ping interval of 1 second before. However, this needs to be further analyzed in the future.

5.2 Distribution of the Outage Durations

The following plot gives an idea, how the outage durations are distributed.

```
[9]: condense_to = 61
sec_bins_not_working_windows = [td.total_seconds() if td.total_seconds() <␣
 ↪condense_to else condense_to for td in not_working_windows]
plt.hist(sec_bins_not_working_windows, bins=condense_to)
plt.title('Histogram of the Duration of Periods without Connectivity condensed␣
 ↪to %s bins' % condense_to)
plt.xlabel('Duration (s)')
plt.ylabel('Frequency')
plt.grid(True)
plt.show()
```

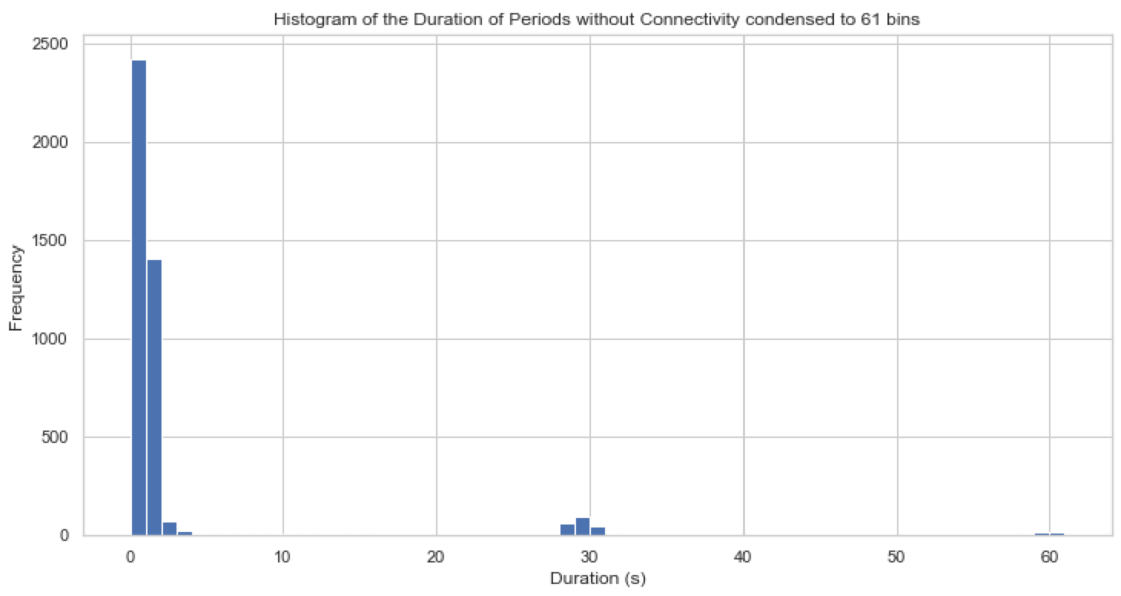


Figure 8: Histogram of the duration of periods without connectivity

This shows that the vast majority of the outage durations are three seconds or shorter. This proves that a Starlink-based Internet connection is generally suitable for a use case without time-critical requirements.

5.3 Latency Behavior after Outages

We also examined the relationship between the length of an outage and the latency immediately after the outage. We assumed that the longer the outage is, the higher the latency immediately afterward is. The bins we used were selected on purpose so that they center around multiples of 15s (see *Additional Findings* for more information on the 15s windows).

[10]:
```
full_not_working_windows = [td.total_seconds() for td in not_working_windows]

bins = [(0, 7), (7, 22), (22, 37), (37, 52), (52, 67), (67, 1000)]
boxes = [None]*len(bins)
outage_counts = [0]*len(bins)

for i in range(len(full_not_working_windows)):
    for b in range(len(bins)):
        if (full_not_working_windows[i] >= bins[b][0] and␣
 ↪full_not_working_windows[i] < bins[b][1]):
            if boxes[b] == None:
                boxes[b] = []
            boxes[b].append(latency_after_not_working[i])
            outage_counts[b] = outage_counts[b] + 1
```

```
labels = ['%s - %s\n%s' % (bins[b][0], bins[b][1], outage_counts[b]) for b in␣
 ↪range(len(bins))]

sns.set(context='notebook', style='whitegrid')
ax = sns.boxplot(data=boxes, showfliers=False)
ax.set_yscale('log')
ax.set(xlabel='Outage Duration (from s - to s)\nNumber of Outages',␣
 ↪ylabel='Latency (ms)')
ax.grid(b=True, which='minor', linewidth=0.5)
plt.xticks(plt.xticks()[0], labels)
plt.title('Distribution of Latencies immediately after Outages')
plt.show()
```

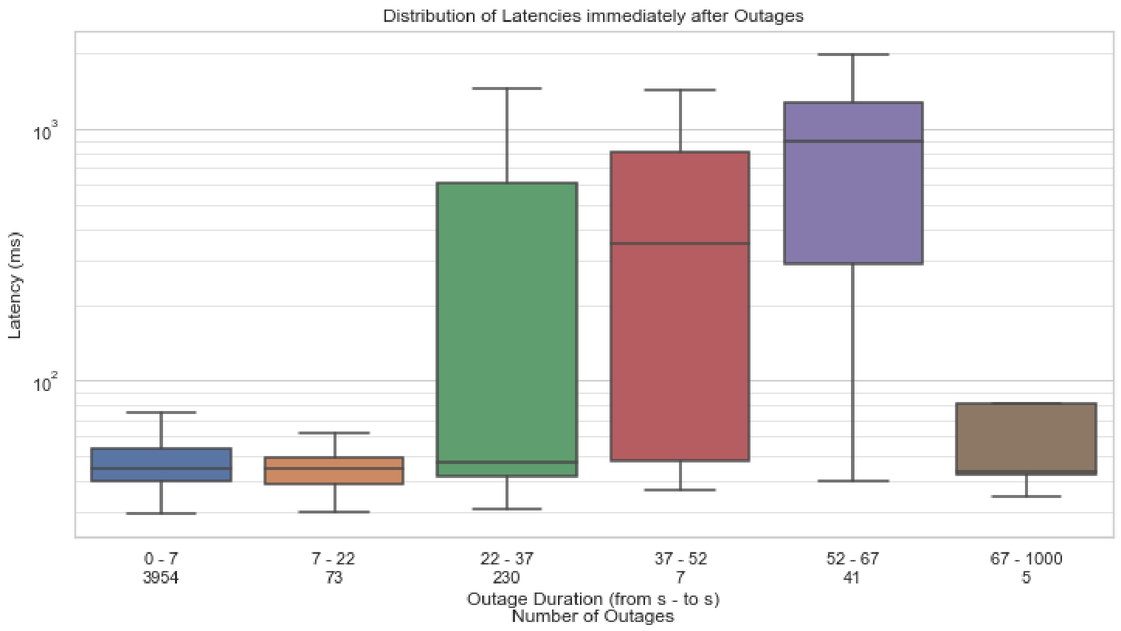


Figure 9: Distribution of the latencies immediately after outages

Our assumptions can be verified from this plot. Due to the very limited statistical relevance of the last block, the fact that it does not match the pattern can be ignored. However, this finding also holds little practical relevance, because a high latency of the first connection after a long outage is not something that a user would be very concerned about.

5.4 Sudden Change in Reliability

The following plot shows a strange behavior that occurred during the period of the test. Starting around noon of 2021-06-04, the number of seconds per hour, when the server was not reachable, decreased and relatively consistently stayed at a low level. As of now, this can not be explained but will be looked into in the future.

```
[11]: unsuccessful_by_hour = {}

      for u in unsuccessful_ts:
          cur_hour = u.replace(minute=0, second=0, microsecond=0)
          if cur_hour in unsuccessful_by_hour:
              unsuccessful_by_hour[cur_hour] = unsuccessful_by_hour[cur_hour] + 1
          else:
              unsuccessful_by_hour[cur_hour] = 1

      plt.scatter(unsuccessful_by_hour.keys(),
                  unsuccessful_by_hour.values())
      plt.title('Long-term Observation of Connection Outages')
      plt.xlabel('Time')
      plt.ylabel('Cumulative Outage Duration (s)')
      plt.grid(True)
      plt.show()
```

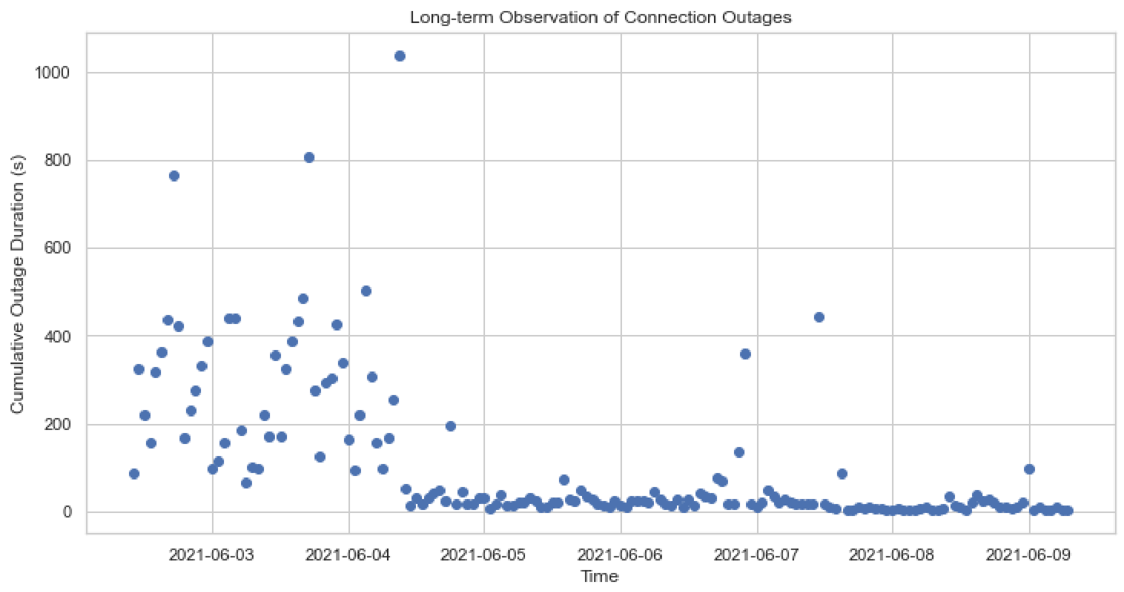
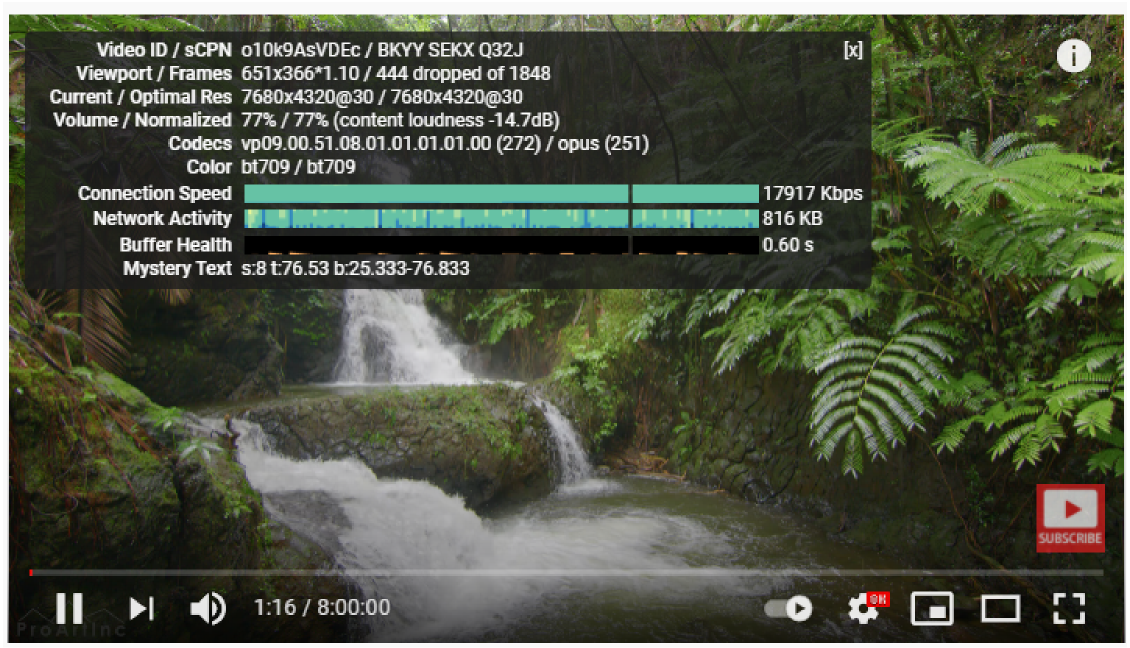


Figure 10: Long-term observation of connection outages

6 Streaming

One of the possible scenarios we want to investigate is how the user experience in terms of constant and stable streaming can be satisfied. We decided to verify this using a standard consumer platform to execute our analysis. To do our verification we did aim for a long-running 8k stream to set up a consumer-grade stream. We decided to extract the important performance indicators using the default YouTube client. This can be seen in the following figure.



```
  Video ID / sCPN  o10k9AsVDEc / BKYY SEKX Q32J
Viewport / Frames  651x366*1.10 / 444 dropped of 1848
Current / Optimal Res  7680x4320@30 / 7680x4320@30
Volume / Normalized  77% / 77% (content loudness -14.7dB)
           Codecs  vp09.00.51.08.01.01.01.01.00 (272) / opus (251)
            Color  bt709 / bt709
 Connection Speed  17917 Kbps
 Network Activity  816 KB
    Buffer Health  0.60 s
     Mystery Text  s:8 t:76.53 b:25.333-76.833
```

Figure 11: A screenshot showing the 8k video and the so-called "stats for nerds" activated

The performance indicators have been acquired from the browser using Optical Character Recognition (OCR) with a sampling rate of 1/2.3s. The measurements described in the rest of this section were executed between 2021-06-08@16:01:54 and 2021-06-09@00:23:34. The measured indicators include:

- Connection Speed (Kbit/s)
- Network Activity (KB)
- Buffer Health (s)

6.1 Overview of the measured Data

Initially, we want to give an overview of the buffer health during the whole measurement. This can be seen in the following figure.

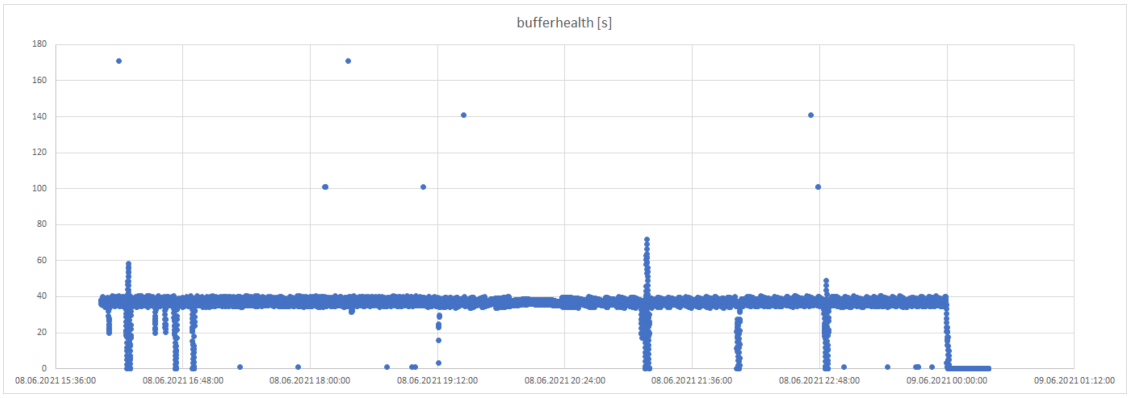


Figure 12: The buffer health of the YouTube client during the whole measurement

The following emerging patterns are worth a discussion and will be examined in the next paragraphs:

1. Singular points due to measurement errors, e.g. at 16:11:34 with a buffer length of 171 seconds.
2. The buffer health of around 40 seconds is relatively constant.
3. The stream was occasionally interrupted, e.g. at 16:43:52.
4. The stream was occasionally delayed, e.g. at 16:06:15. See here.

6.2 Short-term Buffer Drops

In the next figure, we see a drop of the buffer health from 35 seconds down to 20 seconds, but then it recovers again. While recovering we see another drop of buffer size but then it recovers back to 25 seconds right after that. The video stream per se is still steady and there is no negative effect on the user experience.

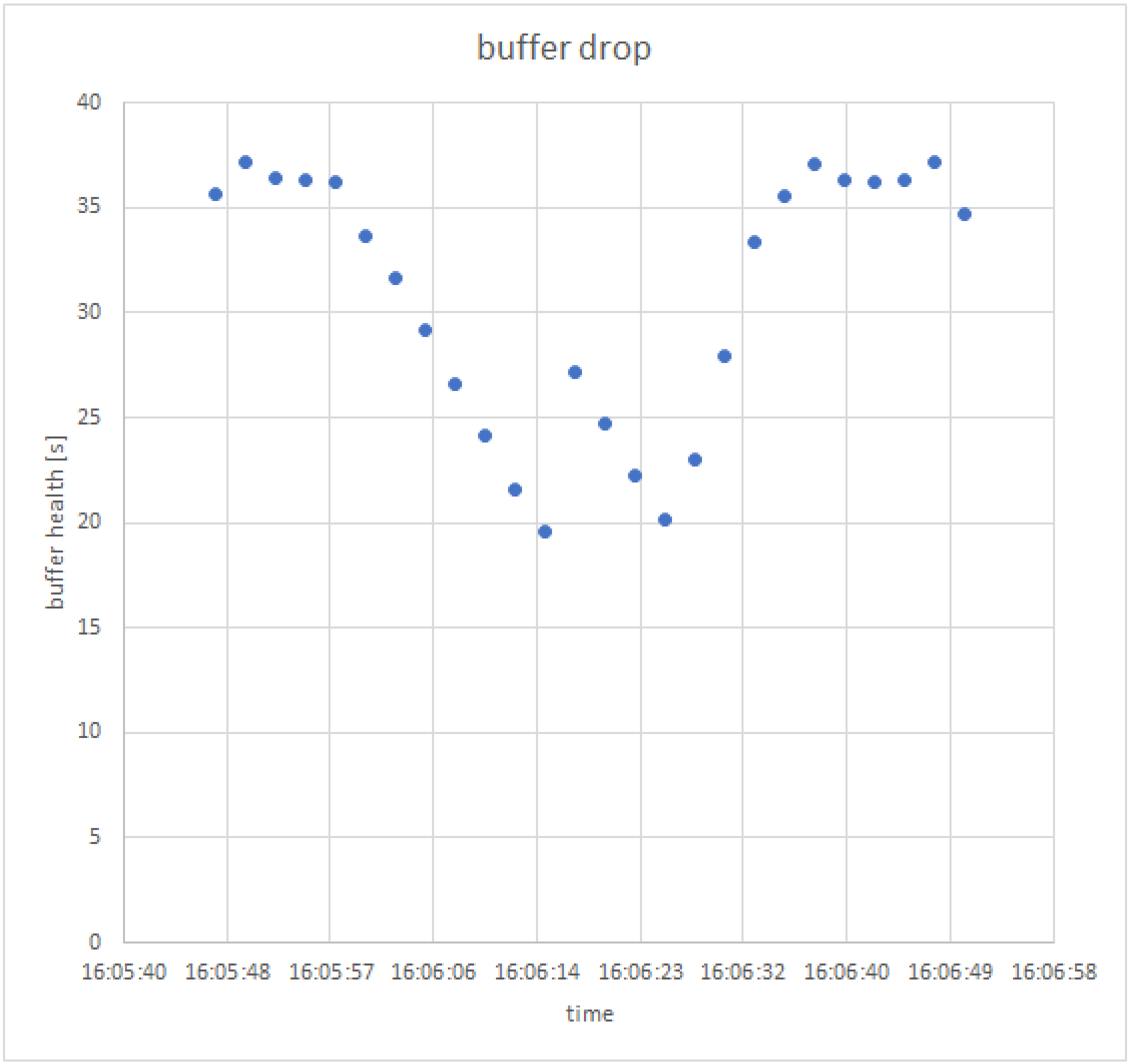


Figure 13: Buffer size drops, but stream recovers

6.3 Longer Buffer Drops and adaptive Buffering

In the next figure, we see a drop in the buffer health from 60 seconds down to 0 seconds which causes the video to stop. The network did fail from 16:17:13 until 16:18:23 but the video stream just failed to play for 4 seconds, due to the extended buffering span of 60 seconds. This buffer size is higher than under usual operations and usually, this is caused by a previous connection problem which causes the web client to increase the buffer size automatically (see here).

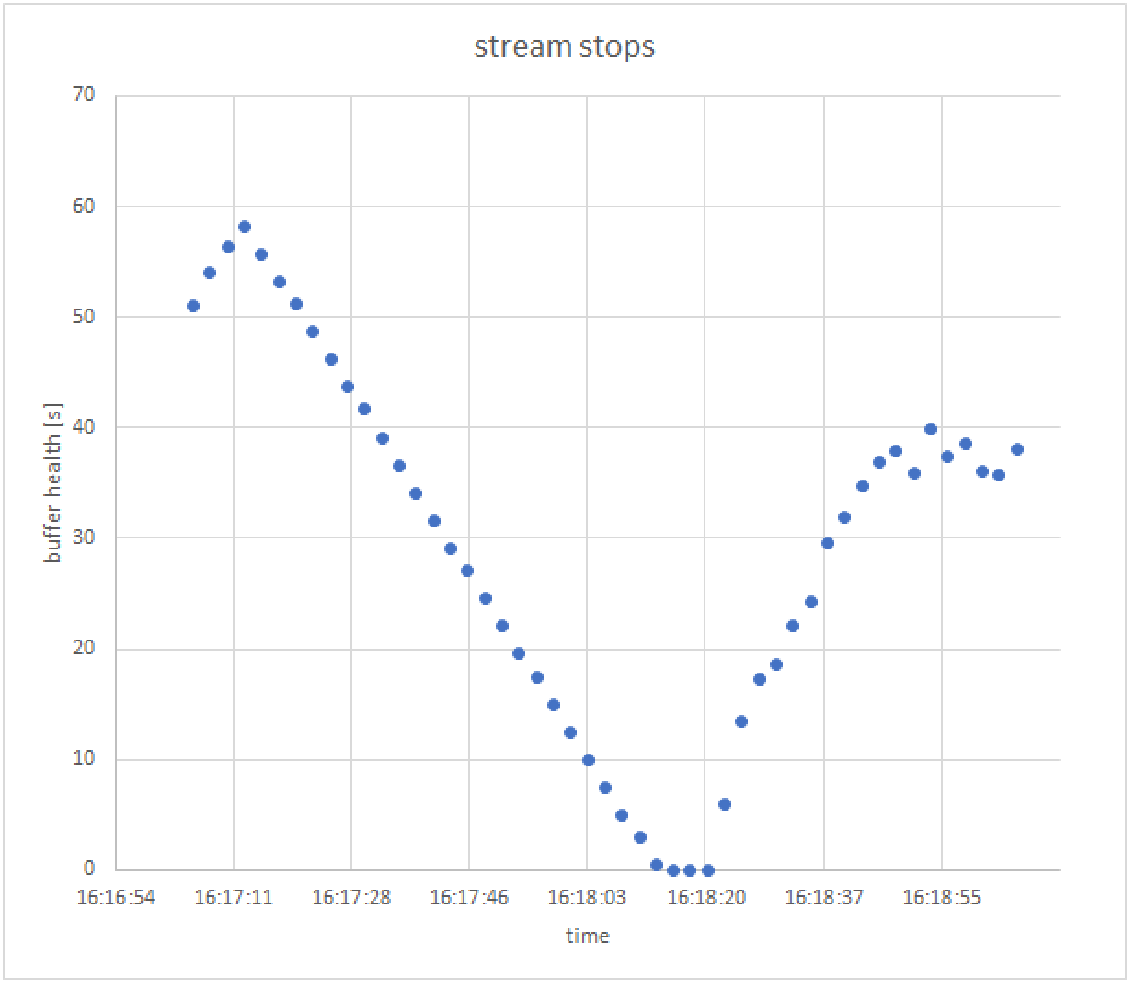


Figure 14: Buffer drop causes streaming to stop for several seconds

6.4 Consecutive Outages

In the next figure, we see a combination of a drop in buffer size followed by two longer connection outages. We also see that the browser client tries to increase the cache size up to 70 seconds. The second outage of the video stream takes around 4 seconds, where the overall connection loss takes place from 21:10:22 until 21:11:40.

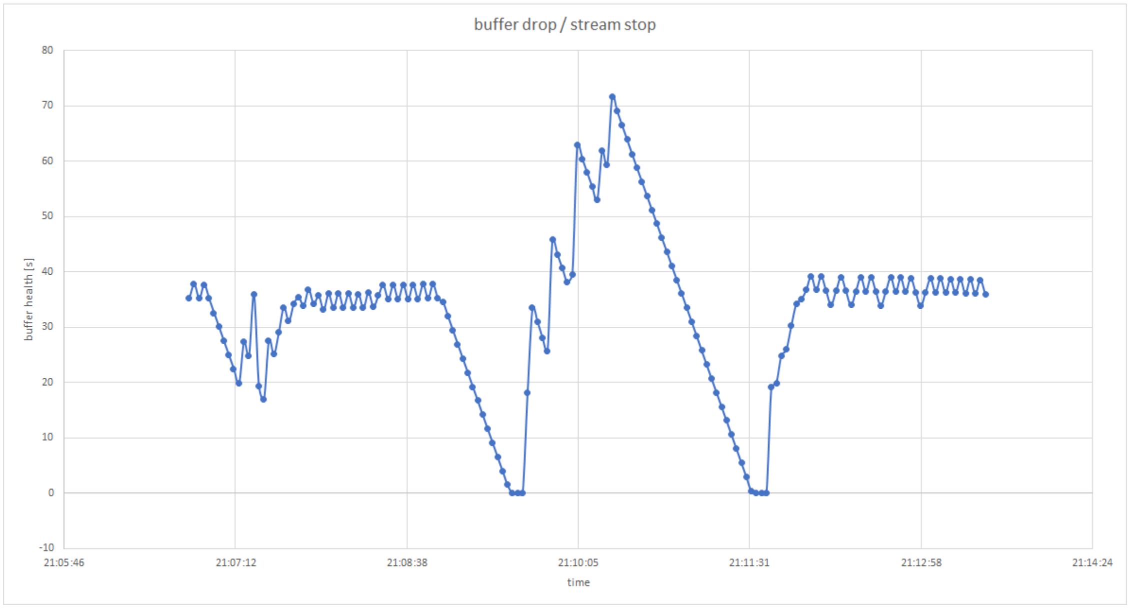


Figure 15: Stream recovers twice after two outages in a row

6.5 Changes in Network Activity

In the next figure, we see the distribution of network activities in KB over time. In general, the browser client does load chunks of data as required to hold a steady buffer size. In the observation window, we see network activities with chunks of up to 5,000 KB, but in the time from 19:12:21 until 22:00:00, we rarely see buffering chunks exceeding 2,000 KB. In general, this would indicate a very steady stream but concerning buffer size, we see a stream collapse at around 21:00:00 which is not represented by bigger video chunks.

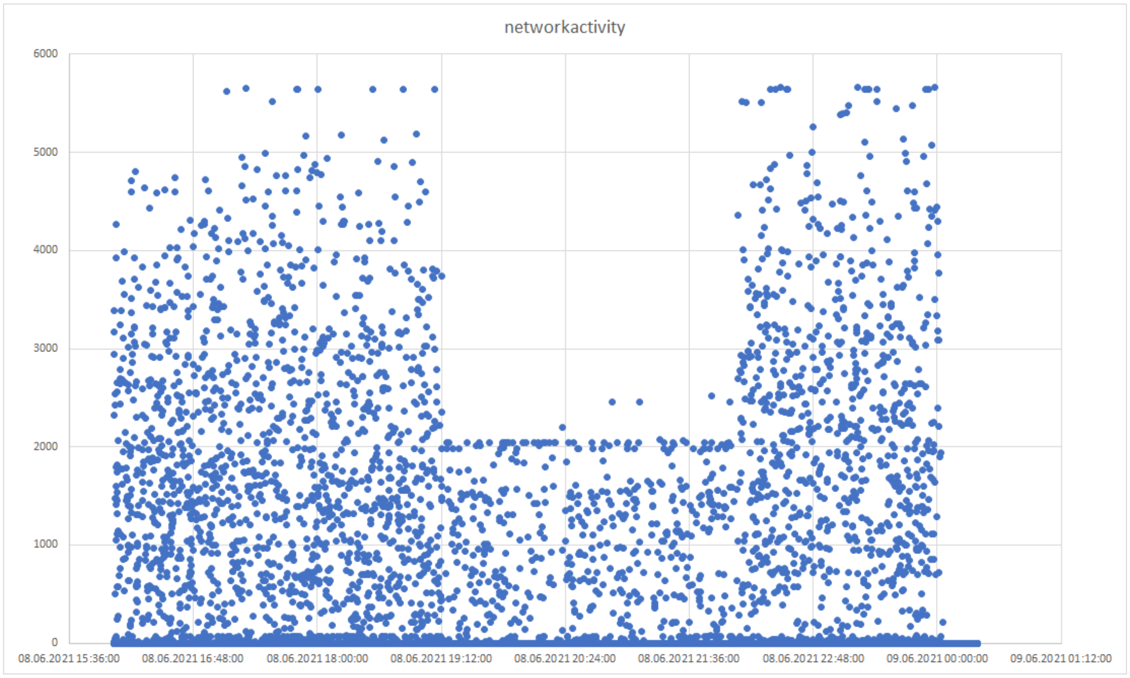


Figure 16: Network activity over time

Unfortunately, we could not find another time frame where the same pattern could be observed. However, at the same time, there also seems to be a change in latencies. Therefore, the following plot observes the time between 2021-06-08@12:00 and 2021-06-09@04:00 in more detail. This also leads toward the conclusion, that the connection was more stable in the said period.

[12]:
```
# defining start and end time, getting the respective indices and processing␣
 ↪the data
start_time = '2021-06-08 12:00:00'
end_time = '2021-06-09 04:00:00'
start_index, end_index = helpers.get_indices_by_time(all_timestamps,␣
 ↪start_time, end_time)
boxes, times, unsuccessfuls = helpers.generate_latency_boxes(all_timestamps,␣
 ↪all_latencies, start_index, end_index)

# creating x-axis labels
labels = []
for time_and_unsuccessfuls in zip(times, unsuccessfuls):
    labels.append('%s\n(%s)' % time_and_unsuccessfuls)

# plotting the data
sns.set(context='notebook', style='whitegrid')
ax = sns.boxplot(data=boxes, showfliers=False)
ax.set(xlabel='Hour\n(number of lost pings)', ylabel='Latency (ms)')
ax.grid(b=True, which='minor', linewidth=0.5)
plt.xticks(plt.xticks()[0], labels)
_p = plt.title('Latency Analysis in Period between %s and %s' %␣
 ↪(all_timestamps[start_index], all_timestamps[end_index]))
```

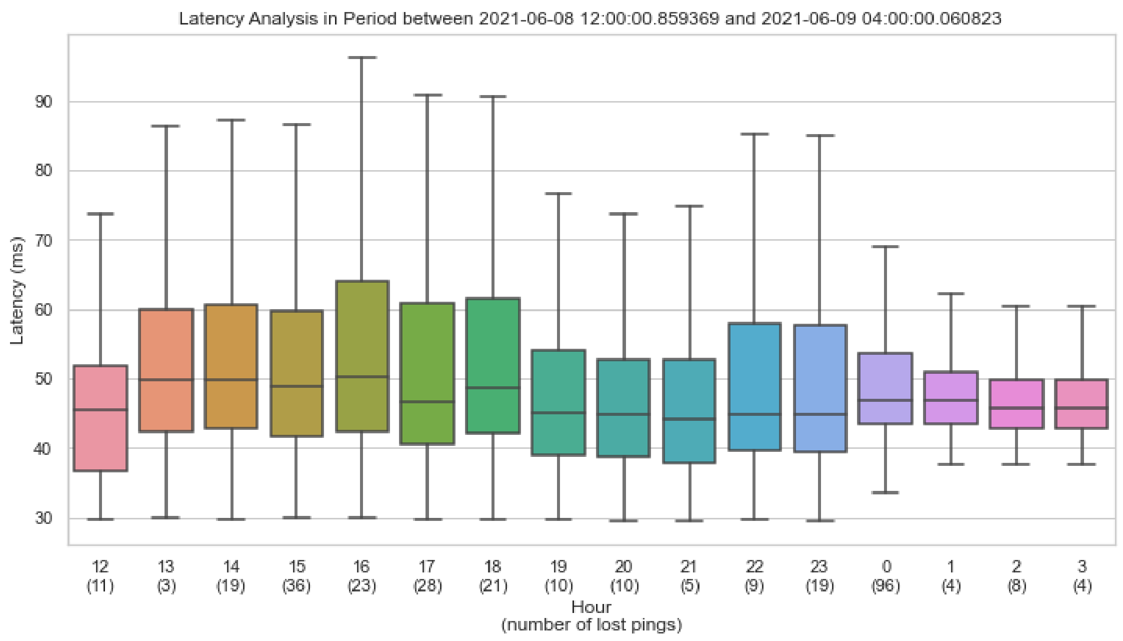


Figure 17: Suspicious change in latencies in the observed period

7 Additional Findings

This section presents additional findings that occurred during the analysis of the gathered data.

7.1 Jumps in Latencies

One of these findings is that there are "jumps" in the latencies, which occur *exactly* at a multiple of 15 seconds (15, 30, 45, ...).

```
[13]: # defining start and end time and getting the respective indices
      start_time = '2021-06-03 08:30:00'
      end_time = '2021-06-03 08:32:00'
      start_index, end_index = helpers.get_indices_by_time(all_timestamps,␣
       ↪start_time, end_time)

      plt.scatter(all_timestamps[start_index:end_index],
                  all_latencies[start_index:end_index],
                  c=colors[start_index:end_index])
      plt.title('Sudden Jumps in the Latencies')
      plt.xlabel('Time')
      plt.ylabel('Latency (ms)')
      plt.grid(True)
      plt.show()
```

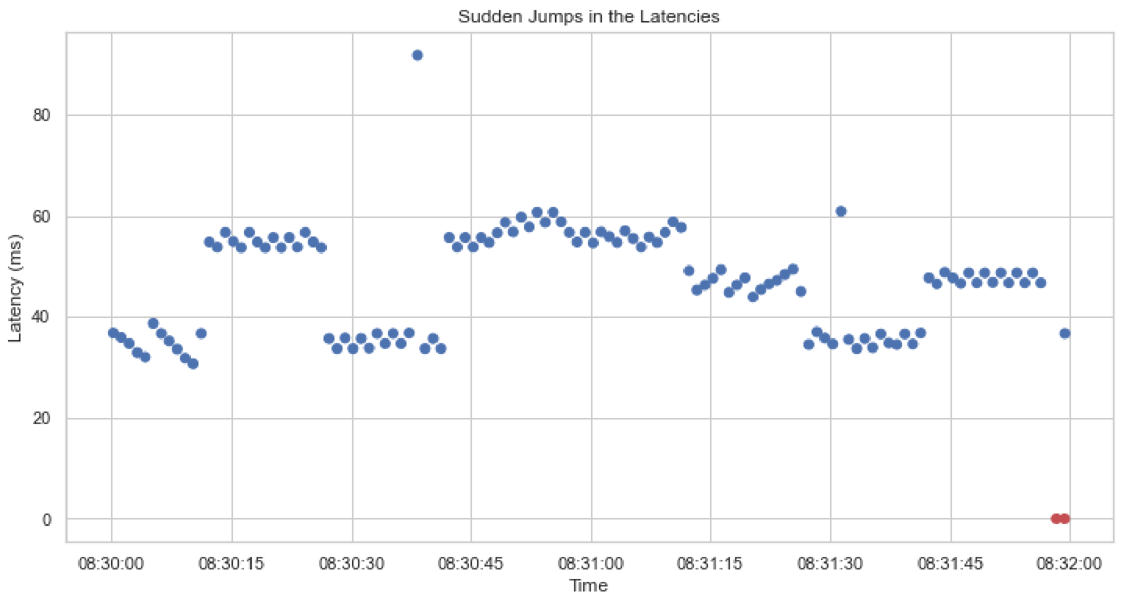


Figure 18: Systematic changes in latency observed

Due to this fact, we also assume that outages likely occur immediately after a period of connectivity that lasted a multiple of 15 seconds. To verify this assumption, the following plot shows how many continuously connected periods there were for the periods from 1 to 69 seconds. The last bar (70) represents the number of continuously connected periods of 70 seconds or more.

[14]:
```
condense_to = 70

sec_bins_working_windows = [td.total_seconds() if td.total_seconds() <=␣
 ↪condense_to-1 else condense_to for td in working_windows]
plt.hist(sec_bins_working_windows, bins=condense_to)
plt.title('Histogram of the Duration of continuously connected Periods␣
 ↪condensed to %s bins' % condense_to)
plt.xlabel('Duration (s)')
plt.ylabel('Frequency')
plt.grid(True)
plt.show()
```

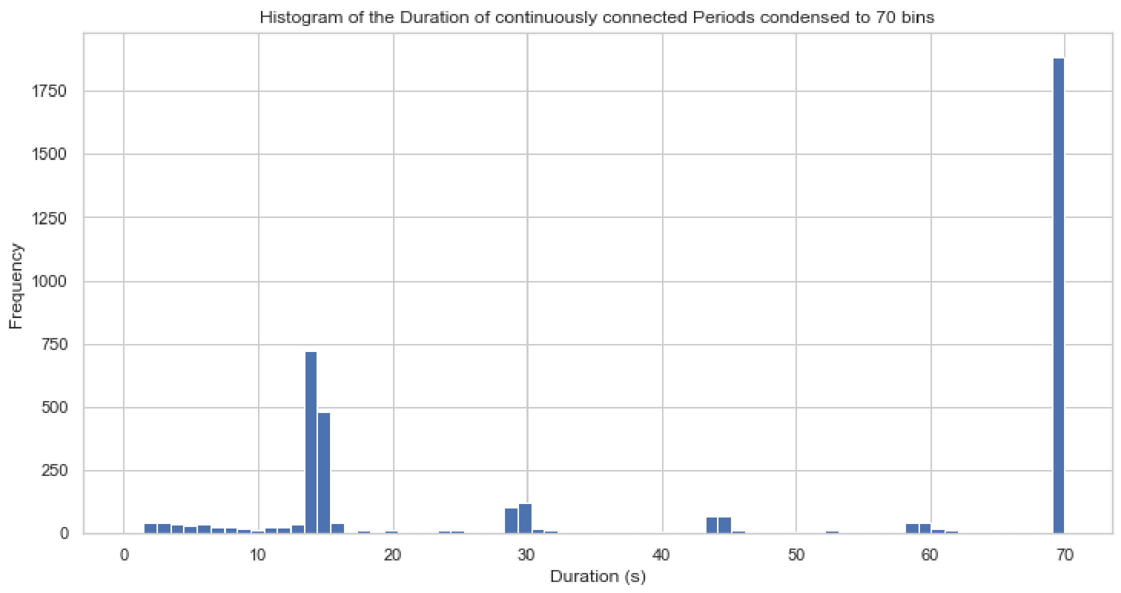


Figure 19: Histogram of the duration of continuously connected periods

This shows that - compared to the surrounding connection periods - we have many connections lasting for around a multiple of 15 seconds, thereby verifying our assumption.

7.2 Satellite Constellations and their Effect

In its current stage, Starlink can not guarantee permanent coverage around the globe. This means that no satellites may be within view of the dish and therefore there is no reception at a given time. To verify this, we decided to make use of the information available via a Live Starlink Satellite

Map. It has to be kept in mind, that this website is not affiliated with Starlink or SpaceX in any way. Therefore, the information gathered there might be incorrect or incomplete.

The data that is loaded in the following code was gathered from said website, which updates the data periodically (every two minutes) and was stored in a log file.

```
[15]: # reading the constellation log file
file = open('constellations_till_202106090619.json')
constellations = json.load(file)
file.close()

# defining start and end time and getting the respective indices
local_start_time = '2021-06-08 13:00:00'
local_end_time = '2021-06-09 08:00:00'
start_ts = datetime.strptime(local_start_time, '%Y-%m-%d %H:%M:%S')
end_ts = datetime.strptime(local_end_time, '%Y-%m-%d %H:%M:%S')

# set start time
constellation_time = start_ts

# initialize variables
times = []
num_of_satellites = []

# iterate over all constellations in the time frame and count how many␣
 ↪satellites are within an assumed visibility window
while constellation_time <= end_ts:
    closest_constellation_index = 0
    closest_time = datetime.
 ↪fromtimestamp(constellations[closest_constellation_index]['stamp'])
    for i in range(len(constellations)):
        current_time = datetime.fromtimestamp(constellations[i]['stamp'])
        if abs(closest_time - constellation_time) > abs(current_time -␣
 ↪constellation_time):
            closest_constellation_index = i
            closest_time = current_time

    home_lat = 46.61
    home_lon = 14.33
    look_north = 6
    look_eastwest = 5

    times.append(constellation_time)
    num_of_satellites.append(0)

    for sat in constellations[closest_constellation_index]['sats']:
        if sat['lat'] > home_lat and sat['lat'] < home_lat+look_north:
            cur_look_eastwest = look_eastwest*((sat['lat']-home_lat)/look_north)
            if sat['lng'] > home_lon-cur_look_eastwest and sat['lng'] <␣
↪home_lon+cur_look_eastwest:
                num_of_satellites[-1] = num_of_satellites[-1] + 1

    constellation_time = constellation_time + timedelta(minutes=1)

plt.plot(times, num_of_satellites, 'rx')
plt.title('Number of assumed visible Satellites between %s and %s' % (start_ts,␣
↪end_ts))
plt.xlabel('Time')
plt.ylabel('Number of Satellites')
plt.grid(True)
plt.show()
```

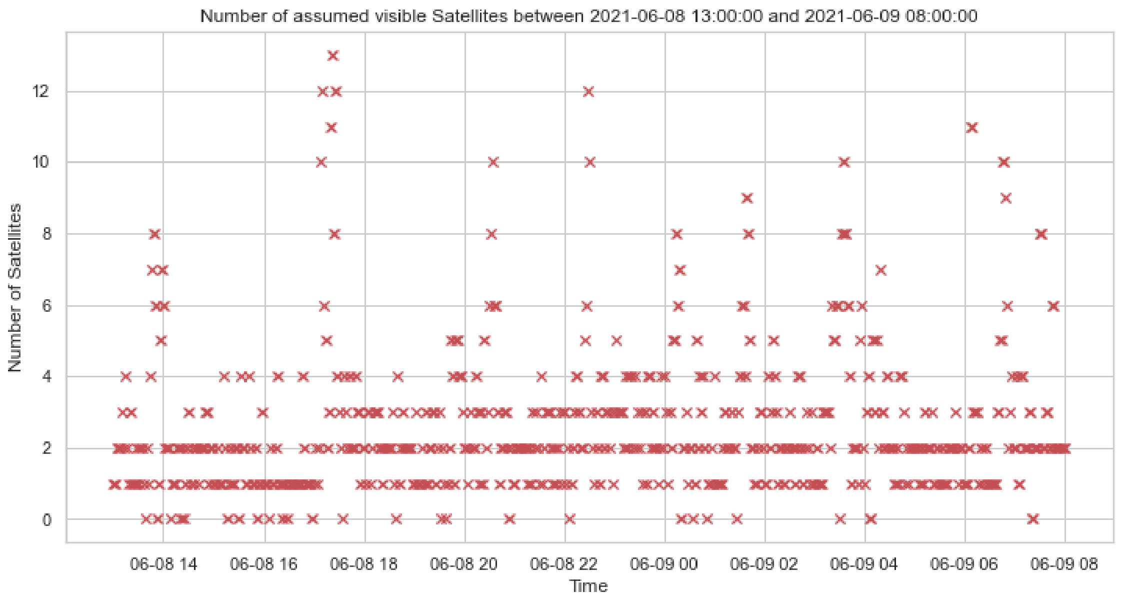


Figure 20: Number of assumed visible satellites in the observed period

This plot shows that there seem to be at least some periods where not a single satellite is within an assumed visibility window (somewhere inside the triangle between Klagenfurt, northwestern Germany, and central Poland. As stated above, this visibility window bases purely on assumptions and needs to be verified in the future!

The next step is to check if there are more lost pings if there are fewer or no satellites inside the assumed visibility window. The checked number of visible satellites is `0`, `1`, `2`, `3`, `4`, and `>= 5`.

[16]:
```
window = 60
ranges = [[0], [1], [2], [3], [4], list(range(5, 20))]

for r in ranges:
    total_count = 0
    actual_outages = 0
    for i in range(len(num_of_satellites)):
        if num_of_satellites[i] in r:
            total_count = total_count + 1
            closest_index = min(enumerate(all_timestamps), key=lambda x:␣
↪abs(x[1]-times[i]))[0]
            if min(all_latencies[closest_index-window:closest_index+window]) ==␣
↪0:
                actual_outages = actual_outages + 1

    print('There were %s instances of seing %s satellites. Of them there were␣
↪%s outages within +/-%s seconds. This is %.0f%%' % (total_count, r,␣
↪actual_outages, window, (actual_outages/total_count)*100))
```

```
There were 52 instances of seing [0] satellites. Of them there were 14 outages
within +/-60 seconds. This is 27%
There were 296 instances of seing [1] satellites. Of them there were 72 outages
within +/-60 seconds. This is 24%
There were 388 instances of seing [2] satellites. Of them there were 99 outages
within +/-60 seconds. This is 26%
There were 180 instances of seing [3] satellites. Of them there were 45 outages
within +/-60 seconds. This is 25%
There were 95 instances of seing [4] satellites. Of them there were 23 outages
within +/-60 seconds. This is 24%
There were 130 instances of seing [5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16,
17, 18, 19] satellites. Of them there were 31 outages within +/-60 seconds. This
is 24%
```

It can be seen that there is no visible correlation between lost pings and visible satellites. Therefore, this theory can be tentatively discarded but should be checked again with more information.

7.3 Influences of the Congestion Windows on Throughputs

The following plot shows that the Transmission Control Protocol (TCP) congestion window significantly influences the throughput. The congestion window is part of the congestion avoidance algorithm built into TCP. Generally speaking, the smaller the congestion window is, the lower is the traffic on the link. The congestion window is dynamically adapted depending on an estimation of how much congestion there is on the link. For this estimation, there are different algorithms available.

```
[17]: file = open('starlink_iperf_tests/iperf-client-upload-20210614.log')
upload_clientside = json.load(file)
file.close()

ul_timestamps_clientside, ul_throughputs_clientside, ul_retransmits_clientside,␣
↪ul_cwnd_clientside = helpers.extract_ts_and_throughput(upload_clientside,␣
↪unit='M')
```

```
ul_timeframe = helpers.get_indices_by_time(ul_timestamps_clientside,␣
 ↪'2021-06-14 20:00:00', '2021-06-14 20:15:00')

plt.rcParams['figure.figsize'] = [12, 9]

fig = plt.figure()
fig.suptitle('Correlation between Congestion Window and Throughput')

ax1 = fig.add_subplot(211)
ax1.scatter(ul_timestamps_serverside[ul_timeframe[0]:ul_timeframe[1]],␣
 ↪ul_throughputs_serverside[ul_timeframe[0]:ul_timeframe[1]], alpha=0.3)
plt.ylabel('Throughput (Mbit/s)')
plt.grid(True)

ax2 = plt.subplot(212, sharex=ax1)
ax2.scatter(ul_timestamps_clientside[ul_timeframe[0]:ul_timeframe[1]],␣
 ↪ul_cwnd_clientside[ul_timeframe[0]:ul_timeframe[1]], alpha=0.3)
plt.ylabel('Congestion Windows (MB)')
plt.grid(True)


plt.xlabel('Time')
plt.show()

plt.rcParams['figure.figsize'] = [12, 6]
```

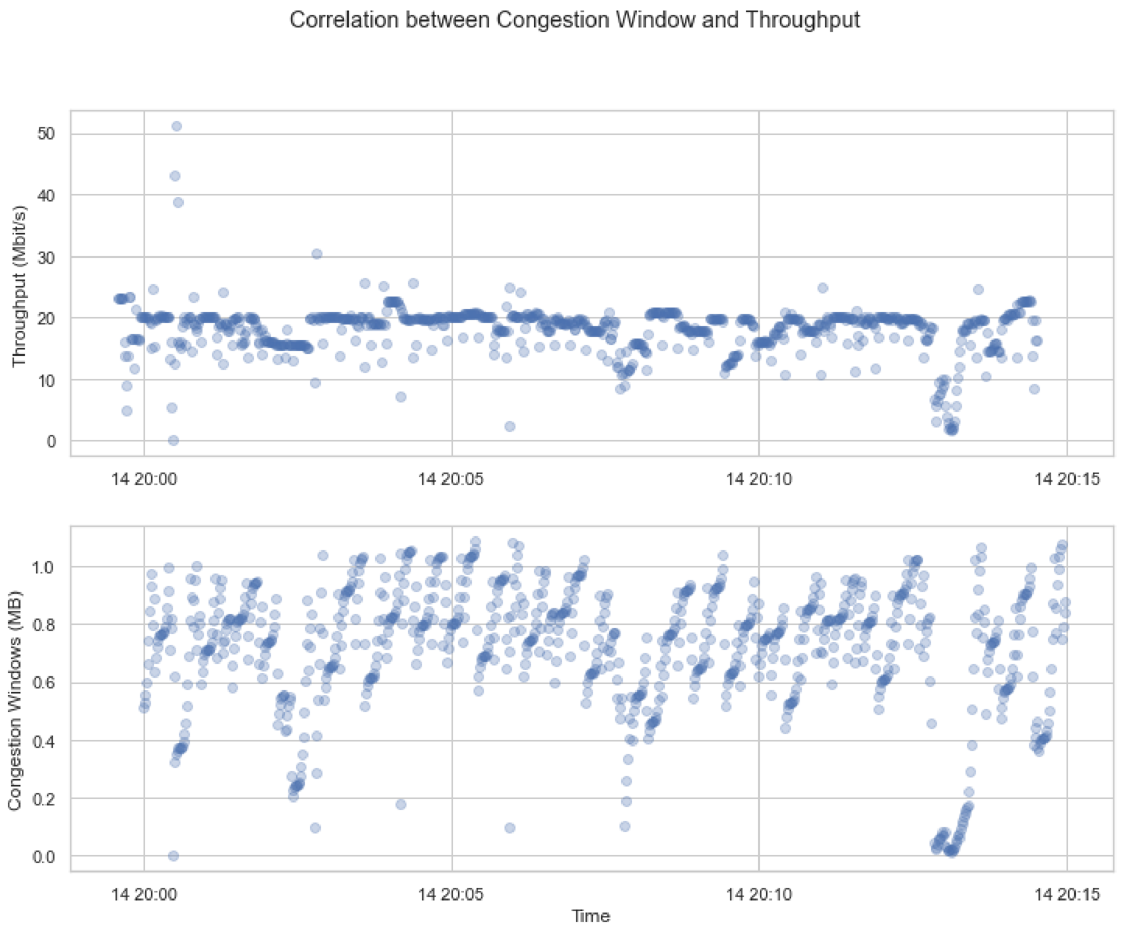


Figure 21: Correlation between congestion window and throughput

7.4 Power Consumption

The power consumption of sending satellite-based systems is known to be significant. It is therefore not surprising, that Starlink is no exception to that. In the observed period of approximately 56 hours, the whole client-side system (router and satellite dish) had a power consumption of 105 Watts on average with a maximum of 190 Watts. On average, this is roughly the same as a state-of-the-art TV.

7.5 Regional Availability

Officially, Starlink Internet access is only offered in selected regions. These regions can be seen on the map below.

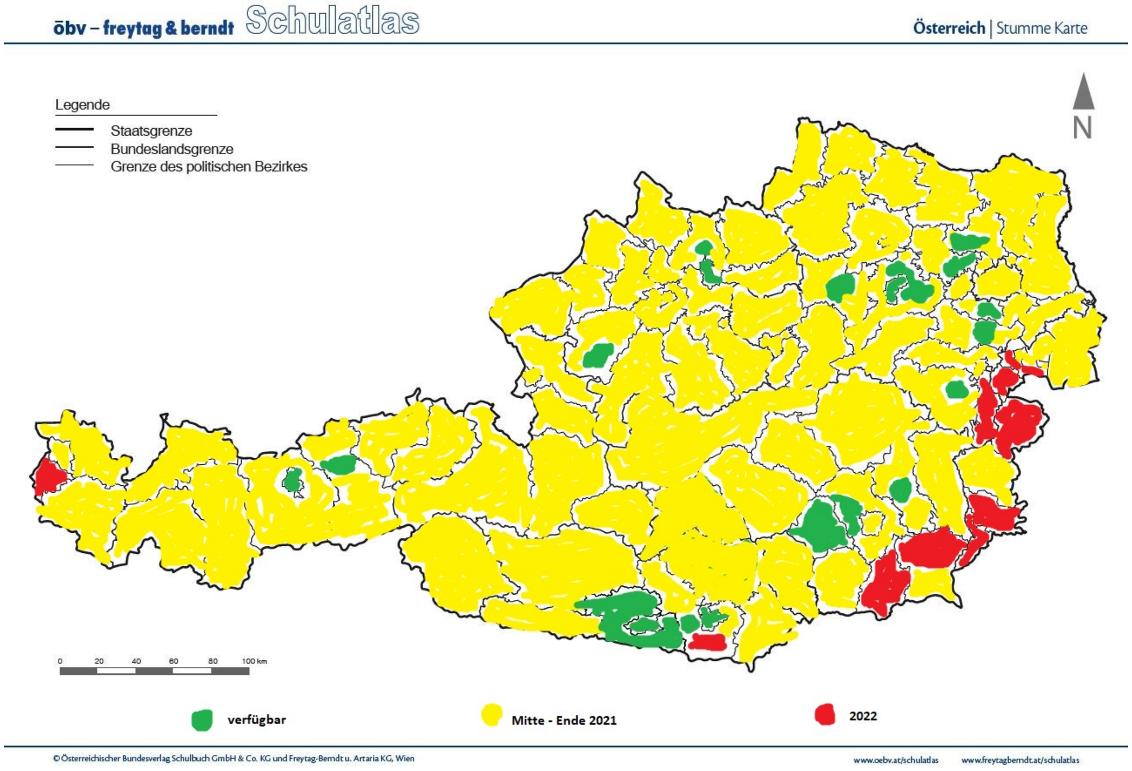

öbv – freytag & berndt Schulatlas
Österreich | Stumme Karte
Legende
Staatsgrenze
Bundeslandsgrenze
Grenze des politischen Bezirkes
N
0 20 40 60 80 100 km
verfügbar
Mitte - Ende 2021
2022
© Österreichischer Bundesverlag Schulbuch GmbH & Co. KG und Freytag-Berndt u. Artaria KG, Wien
www.oebv.at/schulatlas www.freytagberndt.at/schulatlas

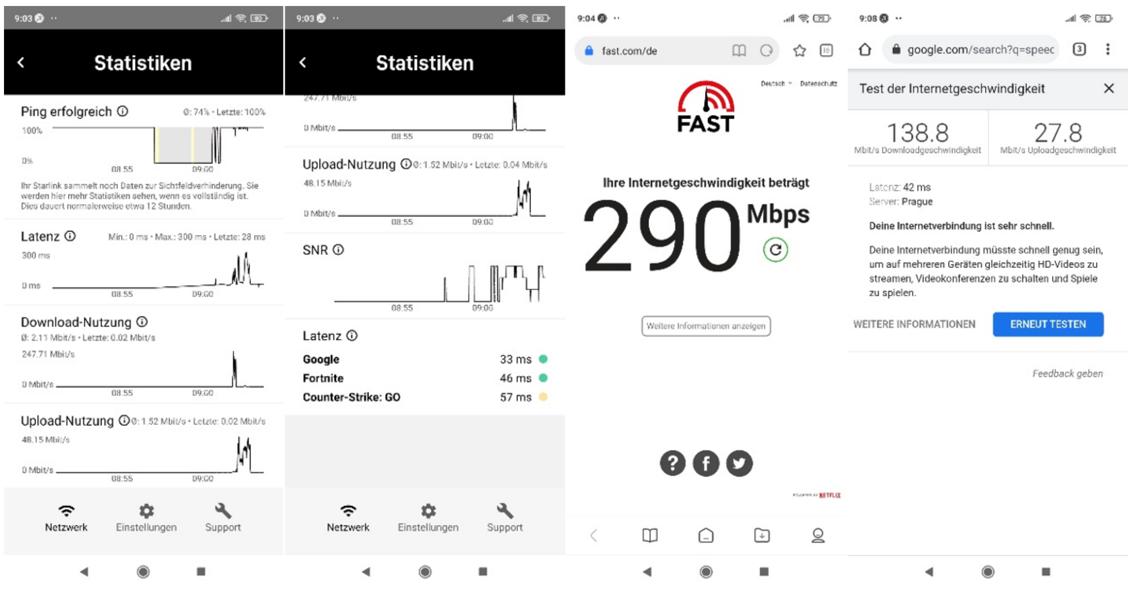
Figure 22: Availability of Starlink-based Internet connections in Austria based on the official ordering system of Starlink

In the green zones, Starlink can be ordered as of now, in the yellow zones, it will be possible to order Starlink by the end of 2021, and in the red zones, Starlink will be available by 2022.

However, we could prove that the connection also works outside these regions at four tested positions. Tested positions include two sites in Klagenfurt (here and here, both in the green zone), one site in Techelsberg (here, in the green zone), and one site in Grafenstein (here, in the red zone). The following screenshots show examples of working connections using the Starlink system in various areas of Carinthia.

7.5.1 Velden am Wörthersee

GPS coordinates: 46.618076, 14.050486



Ping erfolgreich Ø: 74% · Letzte: 100%

Ihr Starlink sammelt noch Daten zur Sichtfeldverhinderung. Sie werden hier mehr Statistiken sehen, wenn es vollständig ist. Dies dauert normalerweise etwa 12 Stunden.

Latenz Min.: 0 ms · Max.: 300 ms · Letzte: 28 ms

Download-Nutzung Ø: 2.11 Mbit/s · Letzte: 0.02 Mbit/s

Upload-Nutzung Ø: 1.52 Mbit/s · Letzte: 0.02 Mbit/s

Upload-Nutzung Ø: 1.52 Mbit/s · Letzte: 0.04 Mbit/s

SNR

Latenz

Google	33 ms 🟢
Fortnite	46 ms 🟢
Counter-Strike: GO	57 ms 🟡

Ihre Internetgeschwindigkeit beträgt 290 Mbps

Test der Internetgeschwindigkeit

138.8 Mbit/s Downloadgeschwindigkeit

27.8 Mbit/s Uploadgeschwindigkeit

Latenz: 42 ms
Server: Prague

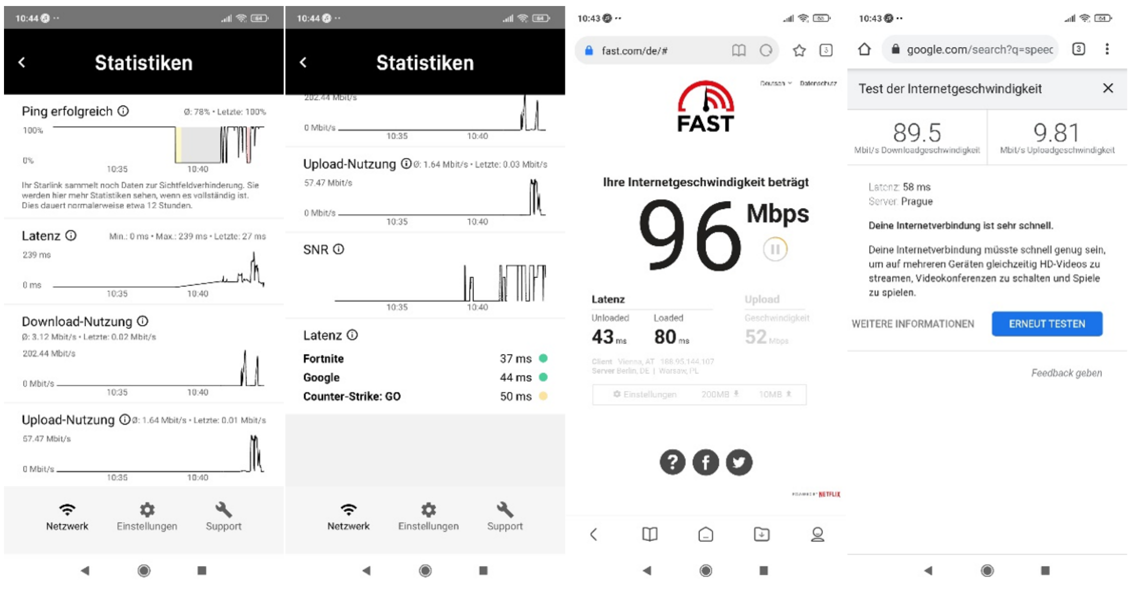
Deine Internetverbindung ist sehr schnell.

Deine Internetverbindung müsste schnell genug sein, um auf mehreren Geräten gleichzeitig HD-Videos zu streamen, Videokonferenzen zu schalten und Spiele zu spielen.

Figure 23: Speedtest result in Velden

7.5.2 Ferlach

GPS coordinates: 46.529411, 14.288259



Ping erfolgreich Ø: 78% · Letzte: 100%

Ihr Starlink sammelt noch Daten zur Sichtfeldverhinderung. Sie werden hier mehr Statistiken sehen, wenn es vollständig ist. Dies dauert normalerweise etwa 12 Stunden.

Latenz Min.: 0 ms · Max.: 239 ms · Letzte: 27 ms

Download-Nutzung Ø: 3.12 Mbit/s · Letzte: 0.02 Mbit/s

Upload-Nutzung Ø: 1.64 Mbit/s · Letzte: 0.01 Mbit/s

Upload-Nutzung Ø: 1.64 Mbit/s · Letzte: 0.03 Mbit/s

SNR

Latenz

Fortnite	37 ms 🟢
Google	44 ms 🟢
Counter-Strike: GO	50 ms 🟡

Ihre Internetgeschwindigkeit beträgt 96 Mbps

Latenz: Unloaded 43 ms, Loaded 80 ms

Upload Geschwindigkeit 52 Mbps

Client Vienna, AT 188.95.144.107
Server Berlin, DE | Warsaw, PL

Test der Internetgeschwindigkeit

89.5 Mbit/s Downloadgeschwindigkeit

9.81 Mbit/s Uploadgeschwindigkeit

Latenz: 58 ms
Server: Prague

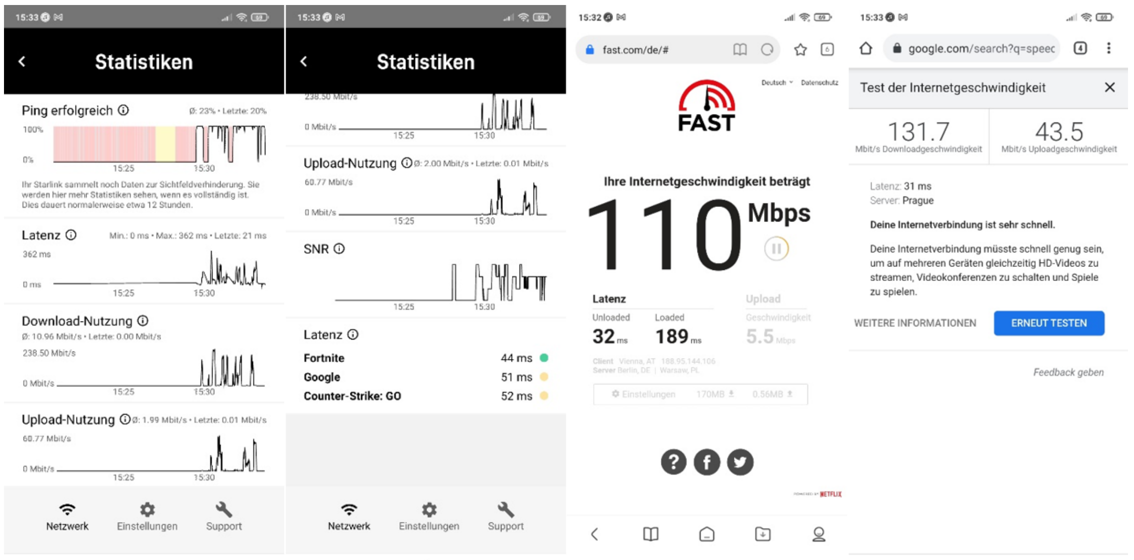
Deine Internetverbindung ist sehr schnell.

Deine Internetverbindung müsste schnell genug sein, um auf mehreren Geräten gleichzeitig HD-Videos zu streamen, Videokonferenzen zu schalten und Spiele zu spielen.

Figure 24: Speedtest result in Ferlach

7.5.3 Feldkirchen in Kärnten

GPS coordinates: 46.744930, 14.048419


15:33

Statistiken

Ping erfolgreich Ø: 23% • Letzte: 20%

Ihr Starlink sammelt noch Daten zur Sichtfeldverhinderung. Sie werden hier mehr Statistiken sehen, wenn es vollständig ist. Dies dauert normalerweise etwa 12 Stunden.

Latenz Min.: 0 ms • Max.: 362 ms • Letzte: 21 ms

Download-Nutzung Ø: 10.96 Mbit/s • Letzte: 0.00 Mbit/s

Upload-Nutzung Ø: 1.99 Mbit/s • Letzte: 0.01 Mbit/s

15:33

Statistiken

Upload-Nutzung Ø: 2.00 Mbit/s • Letzte: 0.01 Mbit/s

SNR

Latenz

Fortnite	44 ms ●
Google	51 ms ●
Counter-Strike: GO	52 ms ●

15:32

fast.com/de/#

Ihre Internetgeschwindigkeit beträgt 110 Mbps

Latenz: Unloaded 32 ms, Loaded 189 ms

Upload Geschwindigkeit 5.5 Mbps

Client Vienna, AT 188.95.144.106
Server Berlin, DE | Warsaw, PL

Einstellungen 170MB 0.56MB

15:33

google.com/search?q=speec

Test der Internetgeschwindigkeit

131.7 Mbit/s Downloadgeschwindigkeit

43.5 Mbit/s Uploadgeschwindigkeit

Latenz: 31 ms
Server: Prague

Deine Internetverbindung ist sehr schnell.

Deine Internetverbindung müsste schnell genug sein, um auf mehreren Geräten gleichzeitig HD-Videos zu streamen, Videokonferenzen zu schalten und Spiele zu spielen.

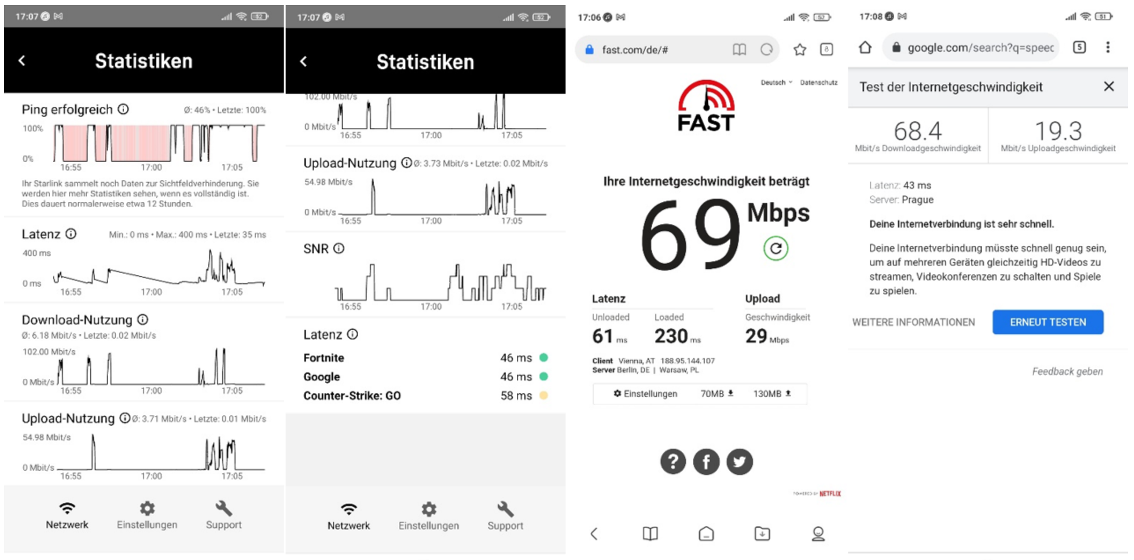
WEITERE INFORMATIONEN ERNEUT TESTEN

Feedback geben

Figure 25: Speedtest result in Feldkirchen

7.5.4 Villach

GPS coordinates: 46.611694, 13.883677


17:07

Statistiken

Ping erfolgreich Ø: 46% • Letzte: 100%

Ihr Starlink sammelt noch Daten zur Sichtfeldverhinderung. Sie werden hier mehr Statistiken sehen, wenn es vollständig ist. Dies dauert normalerweise etwa 12 Stunden.

Latenz Min.: 0 ms • Max.: 400 ms • Letzte: 35 ms

Download-Nutzung Ø: 6.18 Mbit/s • Letzte: 0.02 Mbit/s

Upload-Nutzung Ø: 3.71 Mbit/s • Letzte: 0.01 Mbit/s

17:07

Statistiken

Upload-Nutzung Ø: 3.73 Mbit/s • Letzte: 0.02 Mbit/s

SNR

Latenz

Fortnite	46 ms ●
Google	46 ms ●
Counter-Strike: GO	58 ms ●

17:06

fast.com/de/#

Ihre Internetgeschwindigkeit beträgt 69 Mbps

Latenz: Unloaded 61 ms, Loaded 230 ms

Upload Geschwindigkeit 29 Mbps

Client Vienna, AT 188.95.144.107
Server Berlin, DE | Warsaw, PL

Einstellungen 70MB 130MB

17:08

google.com/search?q=speec

Test der Internetgeschwindigkeit

68.4 Mbit/s Downloadgeschwindigkeit

19.3 Mbit/s Uploadgeschwindigkeit

Latenz: 43 ms
Server: Prague

Deine Internetverbindung ist sehr schnell.

Deine Internetverbindung müsste schnell genug sein, um auf mehreren Geräten gleichzeitig HD-Videos zu streamen, Videokonferenzen zu schalten und Spiele zu spielen.

WEITERE INFORMATIONEN ERNEUT TESTEN

Feedback geben

Figure 26: Speedtest result in Villach

7.6 Quasi-static IP Address Allocation

Throughout our tests, the assigned public IP addresses stayed the same. There seems to be a long-term allocation of the public IP address for each router, which is connected via a dedicated PoP. The IP address that we had assigned during our measurements was 188.95.144.107.

7.7 Varying Time till Connections are established

The time between powering the system up and having a working Internet connection varies between 5 and 20 minutes. The factors that influence that time have not been evaluated so far.

8 Further Information

We want to provide a short list of links that can help out to start some further online research on Starlink and our measurements.

- Satellite constellations can be found at satellitemap.space and starlink.sx.
- Starlink Satellite Dish Analysis can be found at hackaday.com.
- There is an entry about the Starlink Internet infrastructure on reddit.
- It is shown how the Starlink satellite dish looks like disassembled in a teardown on arstechnica.com.