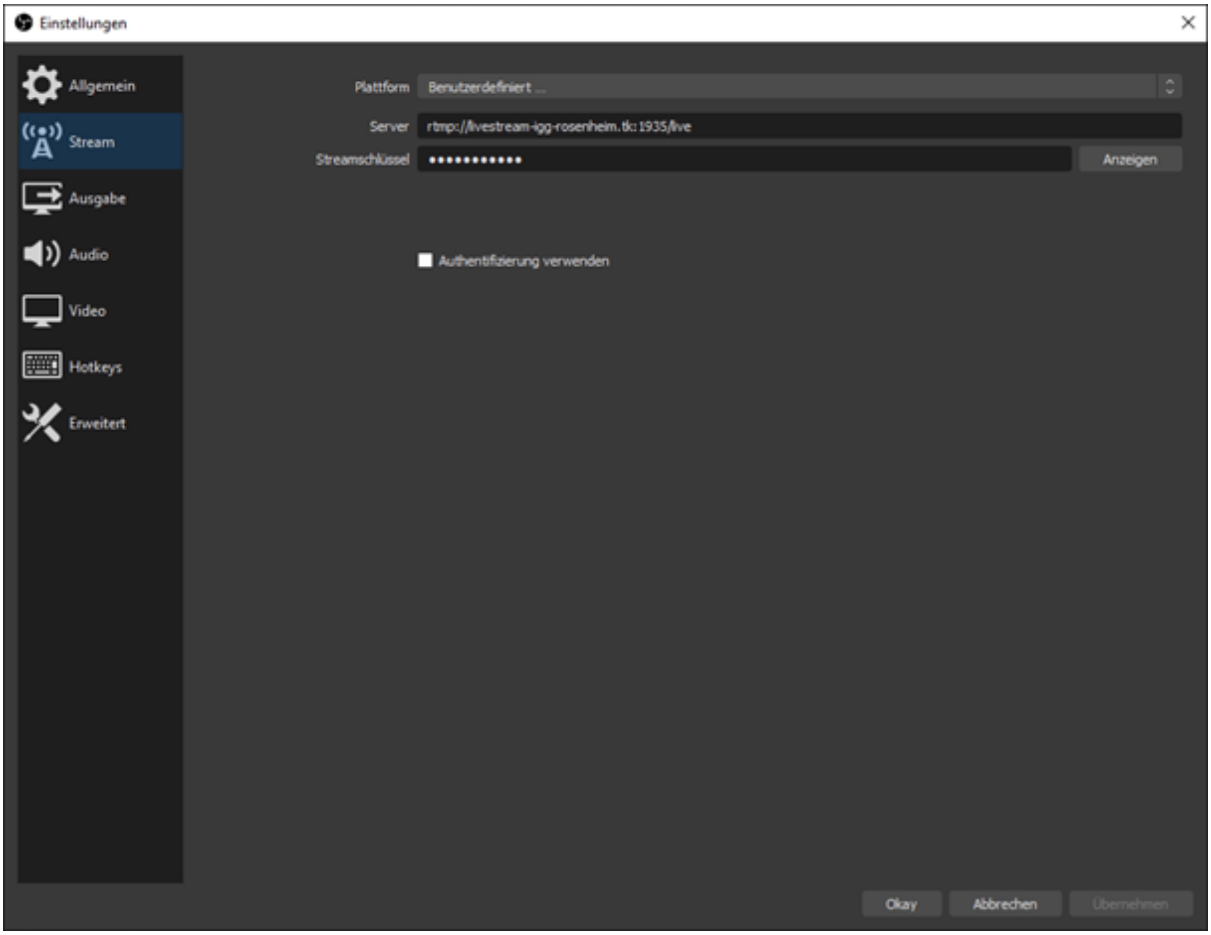
Die Erstellung des Livestreams (auf Linux):

Für unseren Livestream verwendete ich die kostenfreie und open-source Software OBS (https://obsproject.com/).
Hier wird ein Stream über das rtmp-Protokoll (vergleichbar mit dem http-Protokoll, wenn man Websites aufruft) an einen Linux-Server gesendet, welcher diesen empfängt und lokal auf seiner Festplatte speichert.



Zum Umwandeln dieses Protokolles wird das open-source Tool FFmpeg verwendet.
Das FFmpeg-Projekt besteht aus einer Reihe von freien Computerprogrammen und Programm-bibliotheken, die digitales Video- und Audiomaterial aufnehmen, konvertieren, senden (streamen) und in verschiedene Containerformate verpacken können. Unter anderem enthält es mit libavcodec eine umfangreiche Sammlung von Audio- und Videocodecs.

Das Setup des Servers erfolgte folgendermaßen auf einem Debian 10 System:

Benötigte Programme:

- `apache2`
- `ffmpeg`
- `git command line tool`
- `node.js`
- `npm (Node package manager)`

Installation der Programme:

- `sudo apt-get update && sudo apt-get upgrade`
- `sudo apt-get install apache2 ffmpeg git nodejs npm`

Außerdem wird noch ein Third-Party Github-Repository benötigt, um die Verbindung mit dem rtmp-Stream einzugehen.

- `git clone https://github.com/waleedahmad/node-stream.git` (Zur weiteren Installation bitte dort vorbeischauen)

Nun wird selbst programmiert!

- `cd /var/www/html`
- `touch index.html`
- `sudo nano index.html`

index.html

```
<!DOCTYPE html>
  <html lang="de">
  <head>
      <meta charset="UTF-8">
      <meta name="viewport" content="width=device-width, initial-scale=1.0"
>
      <title>IGG Livestream</title>
      <link href="https://vjs.zencdn.net/7.8.4/video-js.css" rel="styleshee
t" />
      <script src="https://vjs.zencdn.net/ie8/1.1.2/videojs-ie8.min.js"></s
cript>
      <link href="https://unpkg.com/[email protected]/dist/video-js.min.css
" rel="stylesheet"/>
      <link href="https://unpkg.com/@videojs/[email protected]/dist/forest/
index.css" rel="stylesheet">

  <style>
      *{
          margin: 0;
          padding: 0;
          font-family: 'Trebuchet MS', 'Lucida Sans Unicode', 'Lucida Grand
e', 'Lucida Sans', Arial, sans-serif;
          outline: none;
      }
      .main{
          margin-top: 5px;
      }
      #viewers{
          padding: 10px 24px;
          color: rgba(88, 88, 88, 0.8);
          font-size: 18px;
          margin-left: 0px;
      }
      #vid1{
          max-width: 600px;
          width: 100%;
          height: 340px;
      }
      .vjs-resolution-button-label, .vjs-resolution-button-staticlabel{
          position:absolute;
          transform:translate(-160%, 100%);
          pointer-events: none;
      }

      </style>
  </head>
    <body>
      <video-js id=vid1 class="video-js vjs-theme-forest" controls>
          <source src="https://livestream-igg-rosenheim.tk/Video/high.m3u8"
type="application/x-mpegURL">
      </video-js>
      <div class="main">
          <h2 id="viewers">Aktuelle Zuschauer: </h2>
      </div>
    <script src="https://vjs.zencdn.net/7.8.4/video.js"></script>
    <script src="https://unpkg.com/@videojs/[email protected]/dist/videojs-
http-streaming.js"></script>
    <script src="videojs-resolution-switcher.js"></script>
    <script>
    var player = videojs('vid1', {
```

```
        controls: true,
        plugins: {
            videoJsResolutionSwitcher: {
              default: 'high',
              dynamicLabel: true
            }
          }
      }, function(){
        // Add dynamically sources via updateSrc method
        player.updateSrc([
            {
              src: 'https://livestream-igg-rosenheim.tk/Video/high.m3u8',
              type: 'application/x-mpegURL',
              label: '1080'
            },
            {
              src: 'https://livestream-igg-rosenheim.tk/Video/low.m3u8',
              type: 'application/x-mpegURL',
              label: '480'
            }
          ])

          player.on('resolutionchange', function(){
            console.info('Source changed to %s', player.src())
            setTimeout(()=>{
                player.play();
            }, 100)
          })

      })

      setTimeout(()=>{
        player.play();
        document.querySelector(".vjs-theme-forest .vjs-control-bar").style.ba
ckground="rgba(57, 130, 191, 0.3)";
        document.querySelector(".vjs-theme-forest .vjs-control-bar").style.ma
rginBottom="0em";
      }, 200)
        var ws = new WebSocket("wss://livestream-igg-rosenheim.tk:8080/")
        ws.onopen = () =>{
            ws.send("_0x33c85f");
        }
        ws.onmessage = ev =>{
            document.querySelector("#viewers").innerText=`Aktuelle Zuschauer:
${ev.data}`
        }

    setInterval(adjustScreenSize, 1000)
    function adjustScreenSize(){
        if(window.innerWidth<600){
            document.getElementById("vid1").style.width="100%"
        }
    }
    setTimeout(adjustScreenSize, 200)
    console.log(document.querySelectorAll(".vjs-menu-item"))
      </script>
      </body>
    </html>
```

Weiter geht's mit der Speicherung des Streams auf dem lokalen Server:

- `sudo mkdir Video`
- `cd Video`

In diesem Video-Ordner wird der Livestream umgewandelt und gespeichert.

- `sudo ffmpeg -i rtmp://EIGENE_DOMAIN/live/EIGENES_PASSWORT -profile:v baseline -level 3.0 -s 1920x1080 -start_number 0 -hls_time 10 -hls_list_size 0 -f hls high.m3u8`

Nun erstellt ffmpeg alle paar Sekunden einen Videoschnipsel der im *.ts Format abgespeichert wird. Mit dem videojs-Player, der oben im HTML benutzt wurde, kann die *.m3u8-Datei nun abgespielt werden. Diese holt sich alle paar Sekunden immer wieder den neuen Videoabschnitt.

Als letztes wird ein Websocket-Server mit node.js aufgesetzt, um die Zuschauer zu zählen:

Hierfür in ein anderes Verzeichnis wechseln:

- `cd ~/Documents`
- `touch server.js`
- `sudo nano server.js`

Um das SSL-Zertifikat zu erhalten, einfach der Anleitung von Certbot folgen:
https://certbot.eff.org/lets-encrypt/debianbuster-apache

(Ein SSL Zertifikat ermöglicht es einer Website, nicht nur unter http besucht werden zu können, sondern auch unter https, da alles noch einmal zusätzlich verschlüsselt wird. Ob eine Website mit diesem Protokoll verschlüsselt ist, kann man sehr gut in der URL-Leiste erkennen. Wenn ein Schloss vorhanden ist, deutet das dies meist darauf hin, dass die Seite verschlüsselt kommuniziert.)

server.js

```
var fs = require('fs');
  var port = 8080;
  // read ssl certificate
  var privateKey = fs.readFileSync('/etc/letsencrypt/live/livestream-igg-rosenheim.tk/privkey.pem', 'utf8');
  var certificate = fs.readFileSync('/etc/letsencrypt/live/livestream-igg-rosenheim.tk/fullchain.pem', 'utf8');

  var credentials = { key: privateKey, cert: certificate };
  var https = require('https');

  //pass in your credentials to create an https server
  var httpsServer = https.createServer(credentials);
  httpsServer.listen(port);

  var WebSocketServer = require('ws').Server;
  var wss = new WebSocketServer({
      server: httpsServer
  });

  var currentViewers = 0;

  wss.on('connection', ws => {
    wss.on('connection', ws => {
      setTimeout(()=>{
        ws.send(wss.clients.size)
      }, 300)
      setInterval(()=>{
          ws.send(wss.clients.size)
      }, 5000);
    })
  })

  console.log(`\nWebsocket Server running on Port ${port}`)
```

Nun den Server starten:

- `node server.js`

Nun sollte alles fertig und bereit zum Streamen sein.

Natürlich wurde hier auf diesen 6 Seiten nur das nötigste besprochen, denn um den Code für den Countdown mit seinen Hover-Effekten einzufügen, wären zusätzliche 3 Seiten vonnöten, welche natürlich zur Hauptsache nichts beitragen, sondern einfach nur eine kleine Spielerei meinerseits sind.

Probleme, die am Tag des Funkkontakts auftraten:

Der Linux-Server verfügt für dieses Event generell über 4 verschiedene offene Ports:

- 80
- 443
- 8080
- 1935

Erstere 3 sind Standardports, welche von fast jeder Firewall und jedem Router durchgelassen werden. Das Ignaz-Netzwerk, welches ursprünglich als Internetanbindung geplant war, erlaubte sogar den vierten. Ein paar Tage vor Beginn des Funkkontakts jedoch bekamen wir die Möglichkeit, eine schnellere Anbindung mit Direktleitung zur Komro zu benutzen. Leider war mir nicht bewusst, dass die Komro über andere Richtlinien bei der Portfreigabe verfügt und erst eine halbe Stunde vor Beginn habe ich gemerkt, dass keine rtmp-Verbindung zum Server möglich ist. Dann begann ein wenig der Kampf gegen die Zeit und durch einen 15-Minütigen Anruf bei der Komro, welche unser Projekt wirklich genial unterstützt hat, konnte ein Techniker mir nach Angabe meiner IP-Adresse Port 1935 freischalten.

Also wichtig wäre für die Zukunft: Unbedingt testen, ob alle Ports bei einem neuen Netzwerk freigeschaltet sind.

Norman Spaeth, P12 2020