Jihočeská univerzita v Českých Budějovicích

Přírodovědecká fakulta

Návrh a implementace platformy síťových služeb Tor založené na Raspberry Pi

Bakalářská práce

Matěj Kouba

Školitel: Ing. Petr Břehovský

České Budějovice 2024

Obsah

1 Cíle **2**

2 Teoretická část **3**

2.1 Rozdělení internetu . . . 3

2.1.1 Surface web . . . 3

2.1.2 Deep web . . . 3

2.1.3 Dark web . . . 4

2.2 Tor . . . 4

2.2.1 Tor prohlížeč . . . 5

2.2.2 Tor uzly . . . 6

2.2.3 Onion služby . . . 10

2.2.4 Torrc soubor . . . 12

2.2.5 Rizika spojená s užíváním a provozováním Tor služeb . . . 13

2.2.6 Nevýhody užívání Toru . . . 15

2.3 Raspberry Pi . . . 16

3 Existující řešení **17**

3.1 tor-relay.co . . . 17

3.2 OnionShare . . . 18

3.3 Tor Control Panel . . . 18

4 Metodika **20**

4.1 Server . . . 20

4.2 Webová aplikace . . . 20

4.3 Hardware . . . 21

5 Výběr hardwaru **22**

5.1 Raspberry Pi . . . 22

5.2 Napájení . . . 23

5.3 Krabička . . . 23

6 Výběr softwaru **24**

6.1 Operační systém 24

6.2 Programovací jazyky a technologie použité v aplikaci 24

6.2.1 Backend 24

6.2.2 Frontend 24

6.2.3 Použité knihovny a framework 24

7 Praktická část **27**

7.1 Instalace 27

7.1.1 Jazyky a knihovny 27

7.1.2 Webový server 28

7.1.3 Tor 28

7.1.4 Mosty 30

7.1.5 Vytvoření přihlašovacích údajů 31

7.2 Struktura aplikace 32

7.3 Funkcionalita aplikace 33

7.3.1 Autentizace 33

7.3.2 Domovská stránka 34

7.3.3 Vytváření nových služeb 34

7.3.4 Limitování Tor služeb 48

7.3.5 Přehled nakonfigurovaných služeb 50

7.3.6 Zapínání/vypínání služeb 55

7.3.7 MyFamily atribut 57

7.3.8 Získávání statusu služby 58

7.3.9 Editování služeb 59

7.3.10 Odstraňování služeb 61

7.3.11 Nahrávání souborů na Onion službu 62

7.3.12 Správa Nginx serveru 64

7.3.13 UPnP 65

7.3.14 Sledování provozu 67

7.3.15 Proxy server 74

7.4 Testování 78

8 Závěr **80**

Seznam použité literatury **81**

Bibliografické údaje

Kouba, M., 1999: Návrh a implementace platformy síťových služeb Tor založené na Raspberry Pi. [Design and implementation of the Tor network services platform based on Raspberry Pi. Bc. Thesis, in Czech.] – 88 p., Faculty of Science, University of South Bohemia, České Budějovice, Czech Republic.

Anotace

This bachelor thesis presents the design and implementation of a user-friendly, Raspberry Pi-based platform for running essential Tor network services (relays, hidden services, and proxies). The platform features a streamlined configuration tool, empowering users to easily set up and manage Tor services for secure communication and anonymous web browsing. This project aims to provide an affordable and accessible solution for privacy-conscious individuals, with a focus on scalability, adaptability, and contributing to the growth of the Tor network.

Prohlašuji, že jsem autorem této kvalifikační práce a že jsem ji vypracoval(a) pouze s použitím pramenů a literatury uvedených v seznamu použitých zdrojů.

V ..... dne..... Podpis .....

Rád bych poděkoval mému vedoucímu práce, panu Ing. Petru Břehovskému, za jeho odborné vedení, cenné rady a trpělivost. Dále bych rád poděkoval mé rodině za podporu v průběhu celého studia.

Přírodovědecká fakulta
Faculty of Science

Jihočeská univerzita v Českých Budějovicích
University of South Bohemia in České Budějovice

ZADÁVACÍ PROTOKOL BAKALÁŘSKÉ PRÁCE

Student:
Matěj Kouba

Program studia/specializace:
Informatika se zaměřením na vzdělávání pro střední školy

Pracoviště PřF JU, kde bude práce vypracována a obhájena:
Katedra informatiky

Školitel:
Ing. Petr Břehovský

Garant z PřF:
doc. Mgr. Lenka Zalabová, Ph.D.

Školitel – specialista, konzultant:
(jméno, příjmení, tituly, u externího š. název a adresa pracoviště, telefon, e-mail)

Téma práce:
Návrh a implementace platformy síťových služeb Tor založené na Raspberry Pi

Cíle práce:

Cílem této bakalářské práce je navrhnout a implementovat platformu založenou na Raspberry Pi, na které lze provozovat různé síťové služby Tor, včetně tor relay, skryté služby a tor proxy. Součástí platformy bude také konfigurační nástroj, který uživatelům umožní tyto služby snadno nastavit a ovládat.

Hlavním cílem tohoto projektu je poskytnout levnou a snadno použitelnou platformu pro ty, kteří chtějí používat Tor pro bezpečnou komunikaci a anonymní prohlížení. Platforma bude navržena tak, aby byla škálovatelná a flexibilní a umožnila uživatelům snadno přidávat nebo odebírat služby podle potřeby.

Kromě technické realizace platformy se tato práce bude zabývat také etickými a právními důsledky používání Tor a dalších nástrojů pro anonymitu. Bude také zkoumat potenciální bezpečnostní rizika a zranitelnosti spojené s provozováním služeb Tor a poskytne doporučení pro zmírnění těchto rizik.

Základní doporučená literatura:
DINGLEDINE, Roger, et al. Tor: The second-generation onion router (2014 DRAFT v1). *Cl. Cam. Ac. Uk*, 2014. https://murdoch.is/papers/tor14design.pdf

POLCÁK, Libor. Základní informace o síti Tor. https://www.fit.vut.cz/research/publication-file/11513/tr.pdf

Tor Project: https://support.torproject.org/

Zvláštní poznámky pracoviště:

Financování práce:

Školitel práce podpis : .....

U externích vedoucích fakultní garant práce podpis : .....

Garant programu/specializace[1][2] podpis : .....

Vedoucí pracoviště PřF JU, kde proběhne obhajoba podpis : .....

Souhlas vedoucího ústavu AV nebo jiné instituce[3] podpis : .....

V Českých Budějovicích dne 17.02.2023

podpis studenta: .....

[1] v případě prací v bakalářském programu Biologie není podpis garanta programu vyžadován
[2] v případě magisterských prací v programech učitelství pro SŠ podpis proděkana pro učitelské obory
[3] v případě, že práce bude vypracovávána jinde než prostorách PřF, například na ústavu AV

Úvod

Právo na svobodný přístup k informacím nám může připadat jako samozřejmost, ale online sledování, cenzura a úniky údajů jej neustále ohrožují. V autoritářských zemích, kde je všudypřítomná cenzura a omezení svobody projevu, se potřeba technologií na ochranu soukromí stává ještě kritičtější. Síť Tor, původně navržená pro americké námořnictvo, představuje mocný nástroj pro novináře, aktivisty a jednotlivce, kteří se snaží chránit své online aktivity a obcházet vládou uložená omezení. Technické složitosti spojené s konfigurací a správou služeb Tor však mohou představovat překážku pro větší rozšíření této sítě.

Tato bakalářská práce si klade za cíl řešit tuto výzvu a přispět k růstu sítě Tor navržením a implementací uživatelsky přívětivé platformy založené na Raspberry Pi pro provozování různých služeb Tor. Platforma bude upřednostňovat snadnost použití s konfiguračním nástrojem, který jednotlivcům umožní zapojit se do sítě bez potřeby rozsáhlých technických znalostí. Nástroj bude navržen tak, aby uživatelům umožňoval přidávat a upravovat služby dle jejich přání.

1. Cíle

- Provést průzkum dostupných řešení pro provozování Tor služeb.
- Vybrat vhodné komponenty a software pro platformu.
- Vytvořit konfigurační nástroj pro nastavování Tor služeb, který bude provozován na Raspberry Pi.
- Integrovat podporu pro různé služby Tor (uzly, mosty, skryté služby, proxy).
- Vytvořit uživatelské rozhraní, který nastavování těchto služeb usnadní.
- Sestavit nástroj tak, aby byl přístupný ze vzdáleného PC.
- Otestovat stabilitu aplikace a samotných tor služeb spuštěných na Raspberry Pi.

2. Teoretická část

2.1 Rozdělení internetu

Pro lepší pochopení toho, co vlastně Tor je, se nejprve podíváme na strukturu internetu. Internet jako celek se skládá ze tří různých vrstev, které se liší nejen obsahem, ale také způsobem, jakým k nim máme přístup. Nyní si podrobněji popíšeme každou z těchto vrstev.

2.1.1 Surface web

První takovou vrstvou je **surface web (povrchový web)**. Tato vrstva zahrnuje stránky, které jsou indexovány a jsou tak přístupné pomocí standardních vyhledávačů jako je Bing, Google nebo Seznam a k jejich přístupu není vyžadován žádný specializovaný software krom klasického internetového prohlížeče.

Některé druhy webových stránek nacházejících se v této vrstvě:

- Zpravodajské webové stránky
- Elektronické obchody
- Vzdělávací zdroje
- Sociální sítě

Dle odhadů tvoří povrchový web pouze 4 % celého internetu[1].

2.1.2 Deep web

Další, největší vrstvou, je **deep web (hluboký web)**. K obsahu deep webu se běžný uživatel nedostane. Je to kvůli tomu, že tato vrstva obsahuje stránky, které nejsou indexovány vyhledávači. Ať už z toho důvodu, že se jedná o webové stránky, které nepoužívají běžné TLD (.com, .cz, .eu, .gov, atd.), nebo indexování záměrně blokují. Pokud se uživatel chce dostat k obsahu deep webu, bude potřebovat příslušnou autorizaci či přihlašovací údaje.

Příklady hlubokého webu:

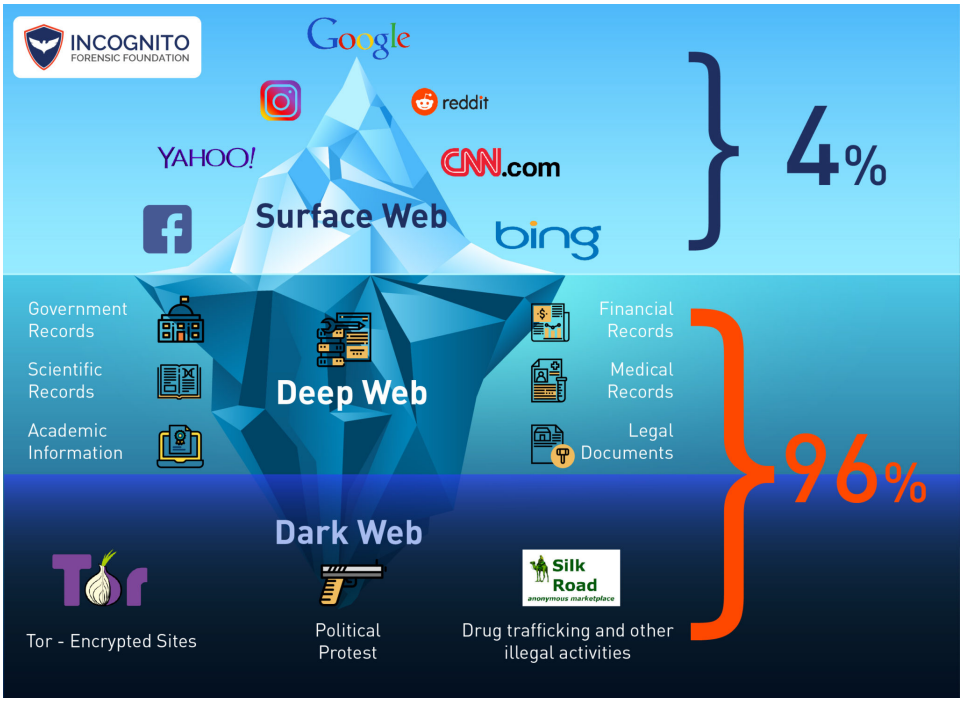
- Emailová schránka
- Cloudové uložiště
- Online bankovní účet

2.1.3 Dark web

Poslední a nejkontroverznější vrstvou je **dark web (temný web)**. Jedná se o část deep webu, která se nachází na takzvaných darknetech, ze kterých se dark web skládá. Mezi darknety se řadí malé sítě mezi přáteli, ale i velké sítě jako je **Tor**, **Freenet** a **I2P**. Pro přístup k těmto sítím je potřeba speciální software. V této práci se budeme soustředit na nejpopulárnější darknet **Tor**, ke kterému se přistupuje pomocí **Tor prohlížeče**[2]. **Dark web** je mezi širokou veřejností znám hlavně díky nelegálním aktivitám, které se na této části internetu nachází. Můžeme zde najít online tržiště, kde prodávající nabízejí zbraně, návykové látky, ukradené osobní údaje včetně platebních karet atd.. Je zde také nabízena řada služeb, jako je najmutí hackera či nájemného vraha. Zůstává otázkou, jak moc jsou tyto nabízené služby skutečné. Z tohoto popisu může spoustě lidem přijít, že na dark webu nemá slušný člověk co pohledávat a měli bychom se této části internetu vyhýbat. Ne vše na dark webu je ovšem spojeno s nelegální aktivitou. Dark web je dnes hojně využíván například zpravodajskými službami, investigativci a whistleblowery díky jeho schopnosti ukrývat identitu uživatele. V autoritářských režimech, kde je přístup k informacím značně omezen, může dark web sloužit ke svobodnému přístupu k médiím, ke kterým by se člověk nemohl normálně nijak dostat.

2.2 Tor

Tor (The onion router) je open-source software provozovaný Tor Projektem, který jeho uživatelům umožňuje anonymní komunikaci po internetu. Využívá takzvaný onion routing. Tato technika zapouzdří data do několika vrstev šifrování, které připomínají vrstvy cibule. Odtud právě pochází onen název. Takto zašifrovaná data následně procházejí sérií uzlů, než dorazí ke

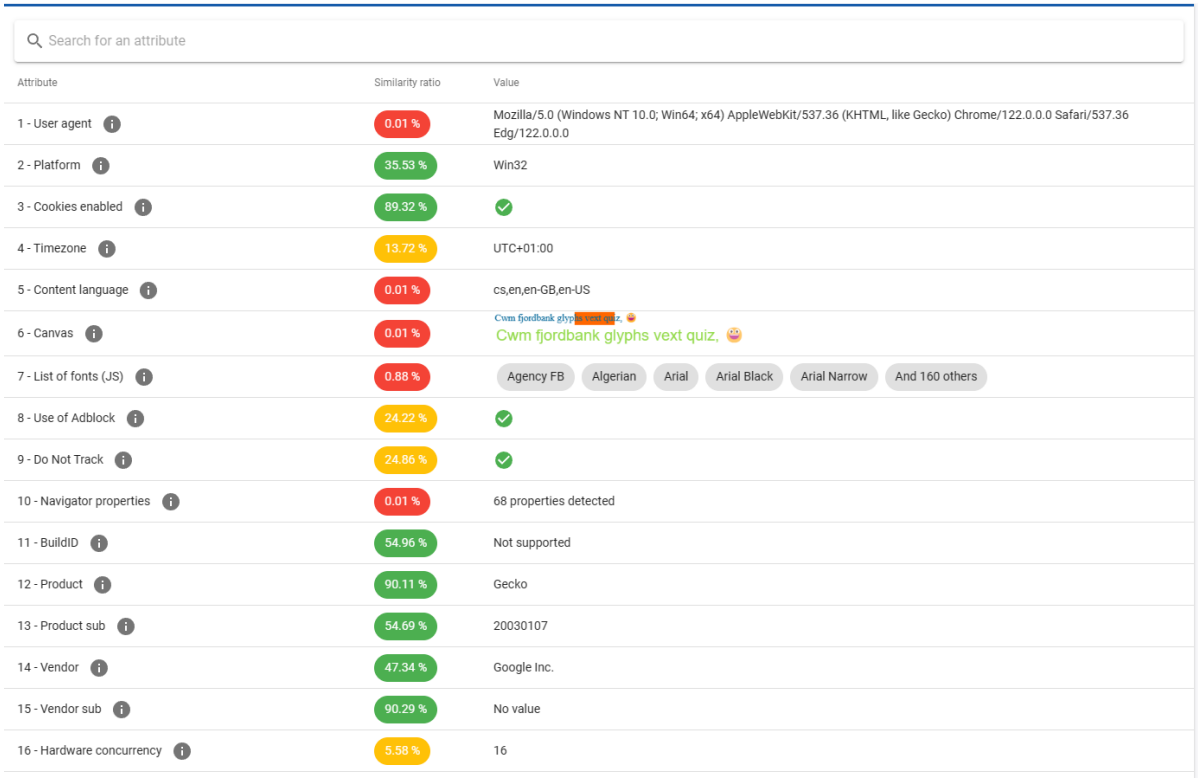


Obrázek 2.1: Vrstvy internetu [3]

své cílové destinaci. Tyto uzly jsou provozovány dobrovolníky po celém světě. Každý uzel odstraní jednu vrstvu šifrování, aby se dozvěděl, jaká je následující destinace, kam data odeslat. Díky tomuto každý z uzlů zná pouze předchozí a následující IP adresu, čímž se zajistí ukrytí skutečné IP adresy uživatele před službou, kam data putují.

2.2.1 Tor prohlížeč

Tor prohlížeč je hlavní způsob, kterým se uživatelé k Toru připojují. Tor prohlížeč je založený na upravené verzi Firefoxu. Krom možnosti připojení k Toru nabízí prohlížeč i další výhody k ochraně soukromý. Pomáhá chránit uživatele před fingerprintingem, který stránky používají k rozlišení jednotlivých uživatelů. Prohlížeč dokáže stránkám poskytnout dostatek informací, díky kterým lze rozlišit jednotlivé uživatele. Na stránce amiunique.org lze vidět, jaké všechny informace prohlížeč poskytuje[4].



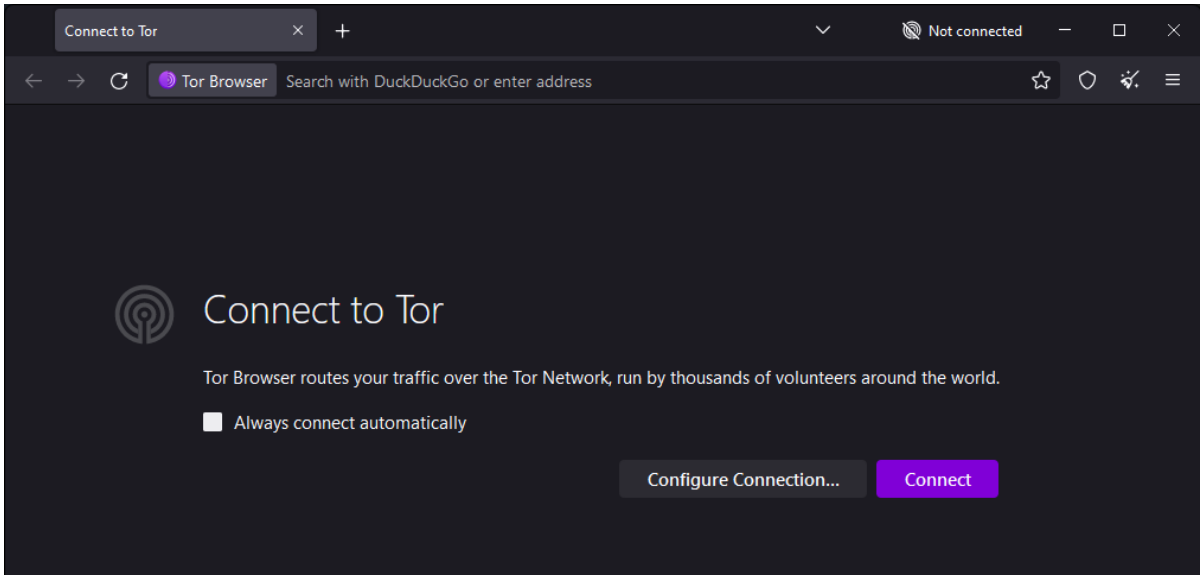
Obrázek 2.2: Výsledek testu na stránce *AmIUnique.org* [4]

Tor prohlížeč se snaží omezit rozdíly mezi zařízeními a operačními systémy maskováním nebo podvržením některých atributů, například platformy, časového pásma a rozlišení obrazovky. V defaultním nastavení také blokuje některé funkce, jako například WebGL, která může odhalit podrobnosti o hardware uživatele. Dále se doporučuje mít nastavený prohlížeč do anglického jazyka a nezvětšovat okno prohlížeče na celou obrazovku, jelikož se jedná o další údaje, díky kterým lze uživatele identifikovat[5].

Prohlížeč defaultně neuchovává historii a soubory cookies jsou platné pouze pro jednu relaci[6].

2.2.2 Tor uzly

Na začátku části o Toru jsme zmínili uzly a nyní si o nich řekneme něco víc. Jak již bylo řečeno, uzly jsou provozovány dobrovolníky po celém světě a dohromady tvoří Tor okruh, který tvoří spojení mezi klientem a sítí Tor. To zajišťuje decentralizaci celé sítě. Uzly se dělí na tři druhy.



Obrázek 2.3: Tor prohlížeč při spuštění

Strážci a prostřední uzly (non-exit uzly)

Strážce je prvním uzlem v řetězci, který tvoří Tor okruh. Prostřední uzel je uzel, který se nachází mezi strážcem a výstupním uzlem.

Výstupní uzel

Jedná se o finální uzel v Tor okruhu, který posílá komunikaci do cíle. Služby, ke kterým se klienti Tor připojují, uvidí IP adresu výstupního uzlu namísto skutečné IP adresy uživatele.

Mosty

Aby síť Tor uměla sestavit okruh, musí být IP adresy všech těchto uzlů veřejně zaznamenány v relay deskriptoru. To bohužel znamená, že poskytovatelé internetových služeb či státní orgány mohou tyto IP adresy, které jsou spojeny s Torem, zablokovat, čímž uživatelům zamezí ve využívání Toru. Z tohoto důvodu existují takzvané mosty. Mosty jsou speciální druh uzlu, který není nikde veřejně zaznamenán a je tím pádem obtížnější je blokovat, protože není na první pohled jasné, že se uživatel připojuje k Toru. Pokud se chce uživatel k Toru připojit prostřednictvím mostu, musí nejdříve o tento most požádat (Většinou prostřednictvím Tor prohlížeče). I přesto se našlo pár zemí, jmenovitě Čína a Írán, které přišly na způsob, jak detekovat připojení pomocí mostů a i ty zablokovat. Z toho důvodu vznikají nové druhy mostů, který každý

funguje na jiném principu.

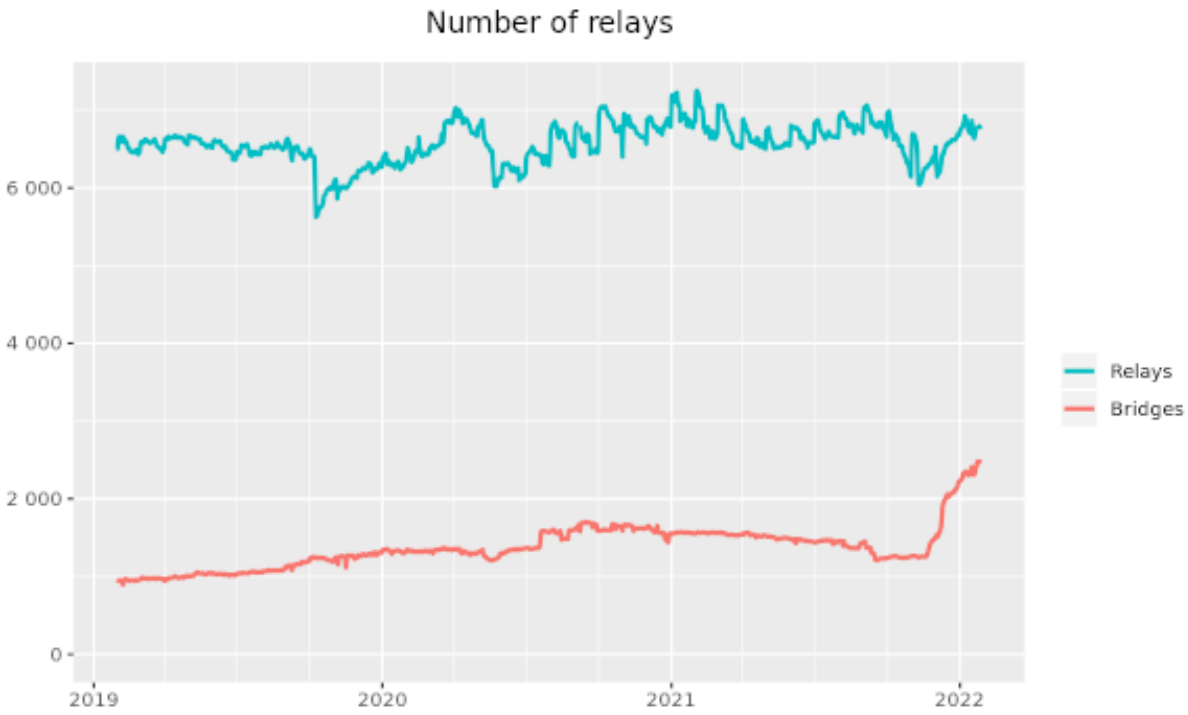
- **Obfs4**: přidává mezi uživatele a most další vrstvu specializovaného šifrování, díky kterému provoz Tor vypadá jako náhodné bajty. Odolává také útokům typu active-probing, kdy cenzor objevuje mosty tím, že se k nim snaží připojit.

- **Meek**: používá techniku zvanou "domain fronting"k odeslání zprávy na Tor relay způsobem, který je obtížné zablokovat. Domain fronting je používání různých doménových jmen na různých komunikačních vrstvách. Program meek-client sestaví speciální požadavek HTTPS a odešle jej zprostředkující webové službě s mnoha doménami za sebou, například CDN. Zvláštností požadavku je, že jeden název domény ("přední doména") se objevuje na "vnější straně"požadavku - v dotazu DNS a SNI - a jiný název se objevuje na "vnitřní straně"- v hlavičce HTTP Host. Cenzor vidí vnější název, ale síť CDN vidí vnitřní název a předá požadavek programu meek-server běžícímu na mostě Tor. Meek-server dekóduje tělo požadavku a předá data do Toru.

- **Snowflake**: Snowflake přesměruje vaše připojení přes proxy servery, aby to vypadalo, že se jedná o videohovor. K připojení se k těmto proxy serverům využívá výše zmíněný domain fronting.

- **WebTunnel**: WebTunnel je nejnovějším přírustkem. Je navržen tak, aby napodoboval šifrovaný webový provoz (HTTPS). Funguje tak, že zabalí spojení payloadu do spojení HTTPS podobného WebSocket, které se pozorovatelům sítě jeví jako běžné spojení HTTPS. Pro pozorovatele tedy vypadá jen jako běžné připojení k webovému serveru, což vyvolává dojem, že uživatel prostě prochází web.

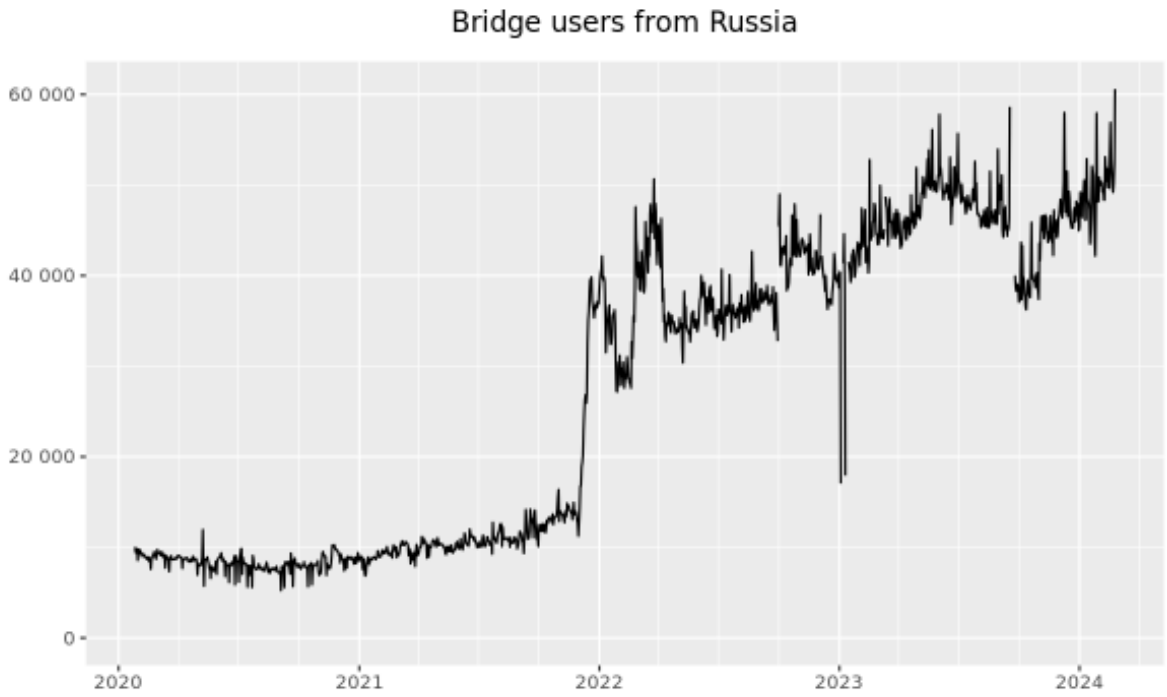
Mosty jsou důležitou součástí Toru, protože umožňují uživatelům z cenzurovaných oblastí přístup k neomezenému internetu. Nejvíce uživatelů mostů pochází z Ruska a kvůli událostem z posledních let se požadavky po mostech neustále zvětšují[9]. Bohužel je mostů značně omezený počet a tak v roce 2021 orgranize Tor Projekt spustila kampaň s názvem "Run a Tor bridge", která měla za úkol rozšířit počet operátorů, kteří mosty provozují. Kampaň skončila úspěchem a počet mostů se podařilo více než zdvojnásobit z 1200 na 2470[7][8].



Obrázek 2.4: Počet mostů po "Run a Tor Bridge"kampani [8]

Country	Mean daily users
Russia	48728 (37.28 %)
Iran	25857 (19.78 %)
United States	15328 (11.73 %)
Germany	4077 (3.12 %)
France	3223 (2.47 %)
United Kingdom	2469 (1.89 %)
Netherlands	2415 (1.85 %)
China	2280 (1.74 %)
India	1688 (1.29 %)
Canada	1213 (0.93 %)

Obrázek 2.5: Podíl uživatelů připojujících se k mostům [9]



Obrázek 2.6: Graf vývoje užívání mostů uživateli z Ruska [9]

2.2.3 Onion služby

Onion služby (dříve známé jako skryté služby) anonymní síťové služby, které jsou vystaveny přes síť Tor. Onion služby nabízejí několik výhod oproti běžným službám:

- **Ochrana IP adres**: Onion služby fungují nad TCP/IP, takže v určitém smyslu nemají IP adresy pro Onion služby význam. V protokolu samotném se ani nepoužívají.
- **End-to-end autentizace**: Když uživatel navštíví některou z onion služeb, ví, že obsah, který vidí, může pocházet pouze z této služby.
- **End-to-end šifrování**: Provoz mezi klientem a onion službou je šifrován podobně jako SSL/HTTPS spojení.
- **NAT punching**: Pokud se vaše zařízení nachází za NATem a nemůžete otevírat porty, může pro vás být obtížné provozovat klasický webový server. Onion služby využívají NAT punching. NAT punching dovolouje klientům přímé peer-to-peer spojení za pomoci známého rendezvous serveru, i když jsou oba klienti za NAT. [10].

Tyto výhody jsou více rozebrány v následujícím popisu protokolu Onion služeb.

.onion TLD

Adresy v TLD onion jsou automaticky generovány na základě veřejného klíče při konfiguraci služby onion. Dříve měly délku 16 znaků pro služby onion V2 [13]. Dnes mají délku 56 znaků pro služby onion V3[13]. Tyto řetězce mohou být tvořeny libovolným písmenem abecedy a desítkovými číslicemi od 2 do 7. Onion adresa v sobě obsahuje veřejný klíč, kontrolní součet a verzi. Všechny tyto údaje jsou zakódovány pomocí base32 [12].
Takto například vypadá .onion adresa na stránku, kterou bychom na internetu našli pod adresou *https://ahmia.fi/*:
juhanurmihxlp77nkq76byazcldy2hlmovfu2epvl5ankdibsot4csyd.onion

Protokol služby Onion

Při připojování se k běžným webovým stránkám se využívá jejich IP adresa. Jak se ale připojit k něčemu, co žádnou IP adresu nemá?

To si nyní vysvětlíme na ukázce toho, jak se navazuje spojení mezi klientem a serverem, který tuto stránku hostí [11]:

1. Onion služba prvně kontaktuje několik Tor uzlů a požádá je, aby fungovaly jako její úvodní body a vytvoří si s nimi spojení. Mezi úvodním bodem a server se nachází další uzly, aby se utajila IP adresa provozovatele onion služby. Služba následně povolí přístup pouze přes tyto body a tím se skryje.

2. Po vytvoření úvodních bodů musí být zajištěn způsob, jak je klienti mohou najít. Z tohoto důvodu se vytvoří deskriptor služby Onion, který obsahuje seznam jejích úvodních bodů a podepíše jej svým soukromým klíčem identity. Zde použitý soukromý klíč identity je soukromá část veřejného klíče, která je zakódována v adrese služby.

3. Služba nahraje tento podepsaný deskriptor do distribuované hašovací tabulky, která je součástí sítě Tor, aby k němu měli klienti přístup. K nahrávání opět používá Tor okruh, aby neprozradila svou adresu či polohu.

4. Ke službě se klient připojí pomocí prohlížeče Tor, který umožňuje připojení k síti Tor. Klient při zadání onion adresy přejde do distribuované hašovací tabulky a požádá o podepsaný deskriptor služby.

5. Po obdržení deskriptoru klient ověří podpis pomocí veřejného klíče. Tím je zajištěna end-to-end autentizace, protože si klient může být jistý, že deskriptor byl vytvořen onou službou, ke které se snaží připojit. Následně klient využije úvodní body uvnitř deskriptoru k tomu, aby se službě představil

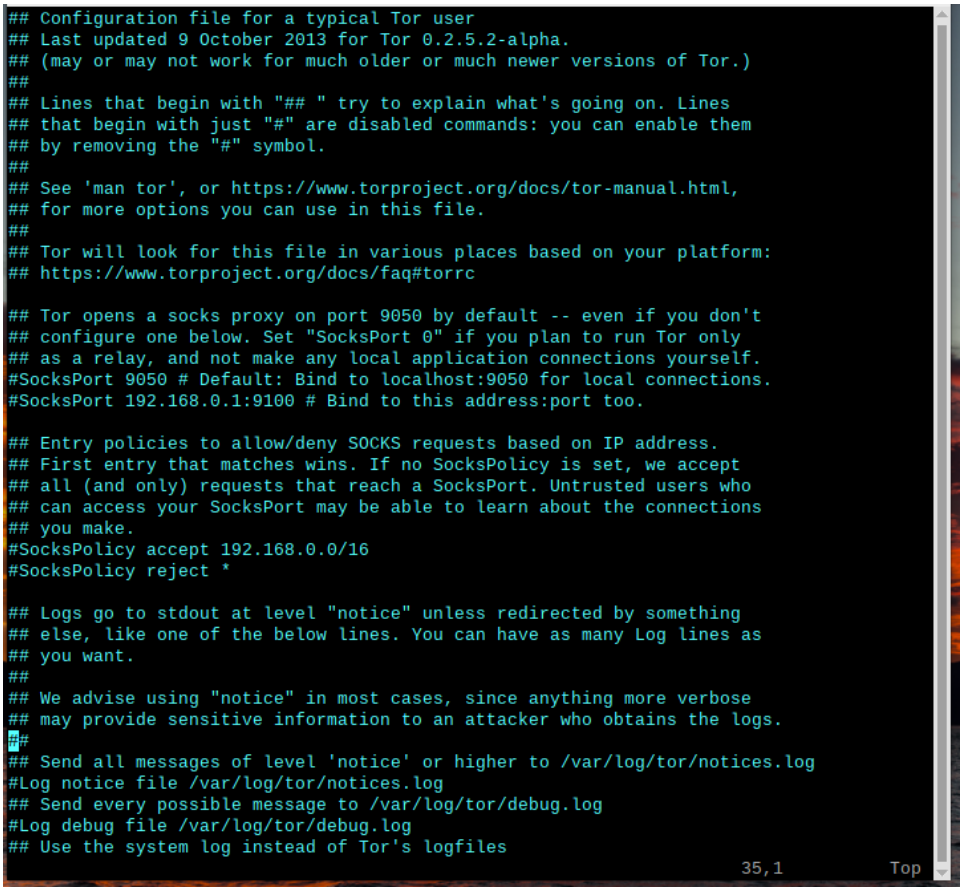
6. Než dojde k představení, klient vybere uzel, který se stane místem setkání mezi klientem a službou a tomuto uzlu předá tajný řetězec, který bude použit při proceduře setkání.

7. Klient pošle úvodnímu bodu adresu setkání a tajný řetězec, které následně úvodní bod předá službě.

8. Služba se přes Tor okruh připojí k místu setkání a odešle mu tajný řetězec. Bod setkání provede konečné ověření toho, že se tajné řetězce shodují. Tento bod poté předává zprávy od klienta službě a naopak. (End-to-end šifrování)

2.2.4 Torrc soubor

Ať už chce operátor provozovat některý z Tor uzlů, onion služeb či mostů, pokaždé se konfigurace provádí přes takzvané **torrc** soubory. Torrc soubory jsou hlavní konfigurační soubory Tor sítě, ve kterých operátor nastavuje vlastnosti a nastavení jím provozované služby. Tomuto souboru a jeho nastavení se více věnujeme v praktické části.



Obrázek 2.7: Torrc soubor

2.2.5 Rizika spojená s užíváním a provozováním Tor služeb

Používání Toru, včetně hostování služeb, není v České republice samo o sobě nelegální, pokud se uživatelé vyhýbají nelegálním aktivitám. Pro Tor v tomto ohledu platí stejné zákony jako pro běžný internet. Jak je to ale s provozem jednotlivých uzlů Toru?

Provozování non-exit uzlů

Všechny IP adresy uzlů jsou veřejně uvedeny v relay-descriptors, takže mohou být zablokovány službami, které neví, jak Tor operuje či je mohou blokovat úmyslně za účelem cenzury uživatele Toru. Pokud máte jednu statickou IP adresu, doporučuje se spíše provozovat most, aby váš provoz mimo Tor nebyl blokován, jako kdyby přicházel z Toru. Pokud máte více IP adres či jednu dynamickou, není to takový problém. Z legálního pohledu operátorovi provozující tento typ uzlů nehrozí žádné větší nebezpečí, protože obsah putující tímto uzlem zůstává vždy v Tor síti.

Provozování exit uzlů

Operátoři exit uzlů mají největší právní odpovědnost. Pokud například uživatel přistoupí k nelegálnímu obsahu při využívání vašeho exit uzlu, všechen tento provoz bude spojen s vaší IP adresou a tím pádem s vámi. Pokud máte v plánu provozovat exit uzel, je dobrý nápad předem kontaktovat vašeho internetového poskytovatele a o této skutečnosti jej informovat. V opačném případě může nastat moment, kdy vám poskytovatel může kvůli stížnostem přerušit připojení k internetu. Tor nabízí na svých stránkách přehled "Good Bad ISPs", kde je uveden seznam těch poskytovatelů internetu, ke kterým má organizace Tor Project informace o jejich postoji k provozu Tor uzlů [23].

Czech Republic

Company/ISP	ASN	Bridges	Relay	Exit	Comments	Last Updated
T-Systems	-	Yes	Yes	?	-	-
M247 Europe	AS9009	Yes	Yes	Yes		04/2023

Obrázek 2.8: Přehled některých českých poskytovatelů internetu a jejich postoj k provozu Tor uzlů [23]

Existují i případy, kdy provozování exit uzlu začala vyšetřovat policie, byť ne přímo v ČR [21][22]. Kvůli legální expozici, která s provozem tohoto uzlu přichází, byste jej neměli provozovat ze svého domova. Exit uzly je dobré provozovat v rámci nějaké instituce. Z těchto důvodů se tímto typem uzlu v praktické části nebudeme zabývat.

Provozování uzlů i mostů v síti Tor s sebou nese také jistou etickou zodpovědnost. Operátor by si měl uvědomit, že jeho uzel může být zneužit k páchání trestné činnosti. Tato práce sice vznikla s myšlenkou rozšíření kapacit sítě Tor s hlavní motivací pro podporu lidí v zemích s omezenou svobodou slova, ale to neznamená, že veškerý provoz přes tyto uzly bude legální. Operátor nemá nad tímto provozem plnou kontrolu. Pokud se operátor s touto myšlenkou nesmíří, raději by se do provozování uzlů pouštět neměl.

2.2.6 Nevýhody užívání Toru

Již jsme si popsali fungování Toru a i některé jeho výhody jako je obcházení cenzury, vylepšení soukromí a přístup na dark web. S užíváním Toru ovšem přichází i pár nevýhod.

- **Rychlost**: Mezi jednu z hlavních nevýhod patří pomalá rychlost. Kvůli směrování provozu přes několik uzlů se výrazně zpomaluje rychlost připojení. Uzly se navíc mohou nacházet daleko od sebe. Proto je Tor nevhodný při činnostech, kdy je požadována vysoká rychlost.

- **Riziko škodlivých uzlů**: Tor funguje skvěle pro skrývání IP adresy, ale neskrývá žádná data, protože přes něj prochází. Pokud tak posíláte data přes HTTP, to jest nešifrovaná, může je provozovatel výstupního uzlu odchytit a následně zneužít. Tomuto útoku se také říká man-in-the-middle útok. Dále útočníci mohou provádět SSL stripping, který zneužívá toho, že uživatelé většinou nepíší celou adresu stránky začínající s "*https://*". Tento útok downgraduje spojení z *https* na *http*, díky čemuž celé spojení přestává být šifrováno. Tor prohlížeč se s tímto snaží bojovat pomocí HTTPS-Only módu.

- **Blokování stránek**: Některé stránky rozpoznávají provoz z Toru a ten automaticky blokují.

- **Blokování skriptů**: Tor prohlížeč blokuje některé skripty. To se týká hlavně sledovacích skriptů a javaskriptů. Takto zablokované skripty mohou znemožnit fungování některých webových stránek či jejich částí.

Pokud jsou tyto nevýhody pro uživatele moc velká překážka a zároveň mu jde pouze o soukromí a nevyžaduje přístup na Dark web, může být vhodnou alternativou použití VPN. VPN je oproti Toru rychlejší, jednodušší na použití a není tak často blokována službami jako Tor. Nevýhodou VPN může být cena, jelikož většina VPN služeb vyžaduje předplatné. Dál musí uživatel důvěřovat VPN poskytovateli, kterého si zvolí, protože může vidět a zaznamenávat provoz uživatele.

2.3 Raspberry Pi

Raspberry Pi je série malých jednodeskových počítačů vyvíjená společností **Raspberry Pi Ltd.**. Vše potřebné pro plně funkční počítač - například procesor, paměť, grafický čip a úložiště - je zabudováno na jediné desce plošných spojů. Tato zařízení využívají procesorů s architekturou ARM. Kromě své kompaktní velikosti se Raspberry Pi vyznačuje také nízkou cenou a nízkou energetickou spotřebou. Navzdory těmto vlastnostem poskytují dostatečný výkon, což umožňuje široké využití těchto počítačů od hraní nenáročných her, přes výuku programování až po automatizaci a robotizaci.

V době psaní této práce existují tři série Raspberry Pi.

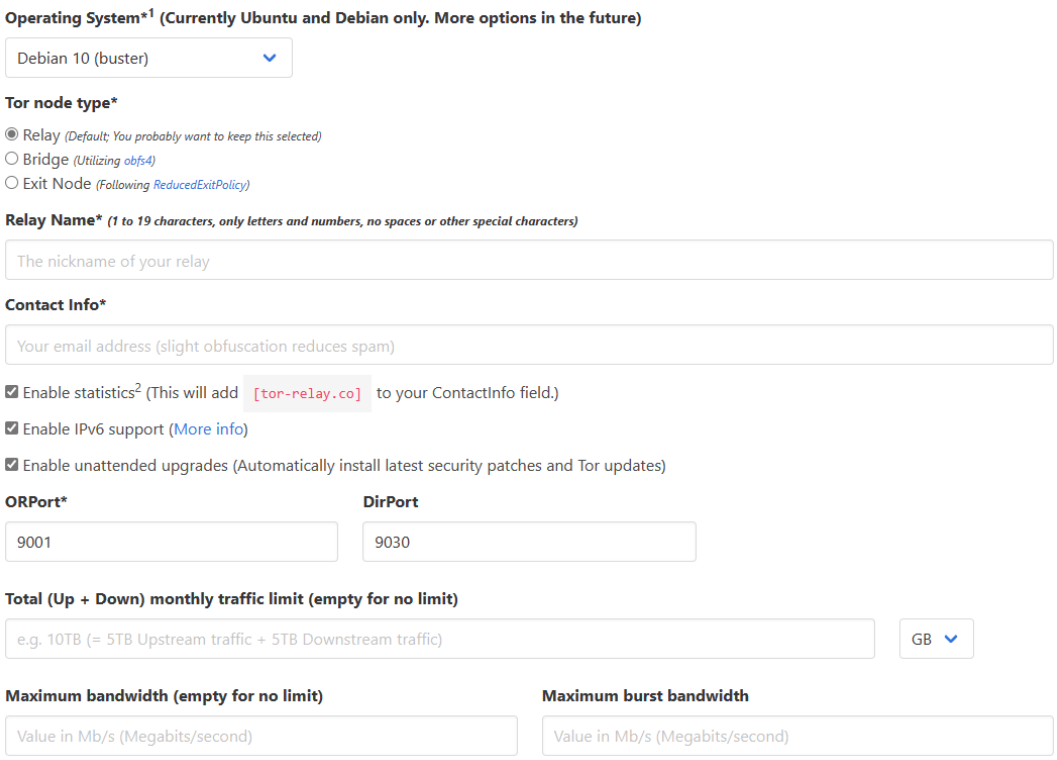
- **Raspberry Pi** - Nejdražší, ale zároveň nejvýkonnější série
- **Raspberry Pi Zero** - Zmenšená, levnější a méně výkonná série
- **Raspberry Pi Pico** - Mikrokontroler pro vytváření vestavěných systémů.

3. Existující řešení

Při rozhodování toho, jaké funkce by měla aplikace obsahovat, byla analyzována již existující řešení pro konfiguraci Tor služeb. V době psaní práce jsem byl schopen naleznout celkem čtyři možnosti. První možností je postupovat klasickou cestou a to je ruční konfigurace *torrc* souborů pomocí textového editoru a následné spouštění toru přes terminál.

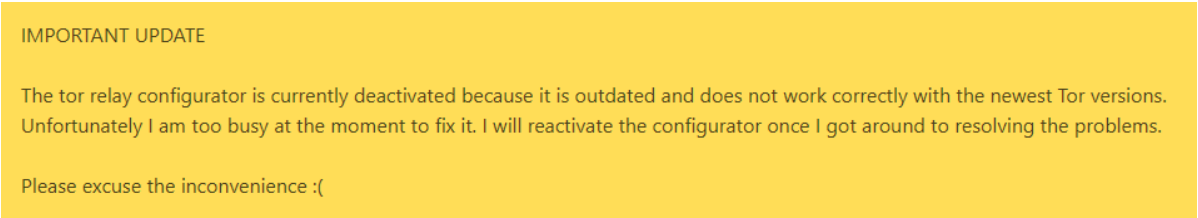
3.1 tor-relay.co

Druhým řešením je stránka *https://tor-relay.co/*, která umožňuje vytvoření konfiguračního skriptu z hodnot, které si uživatel určí. Na této stránce je možno vygenerovat skript pro vytvoření non-exit a exit uzlu a obfs4 mostu.



Obrázek 3.1: *tor-relay.co*[**?**]

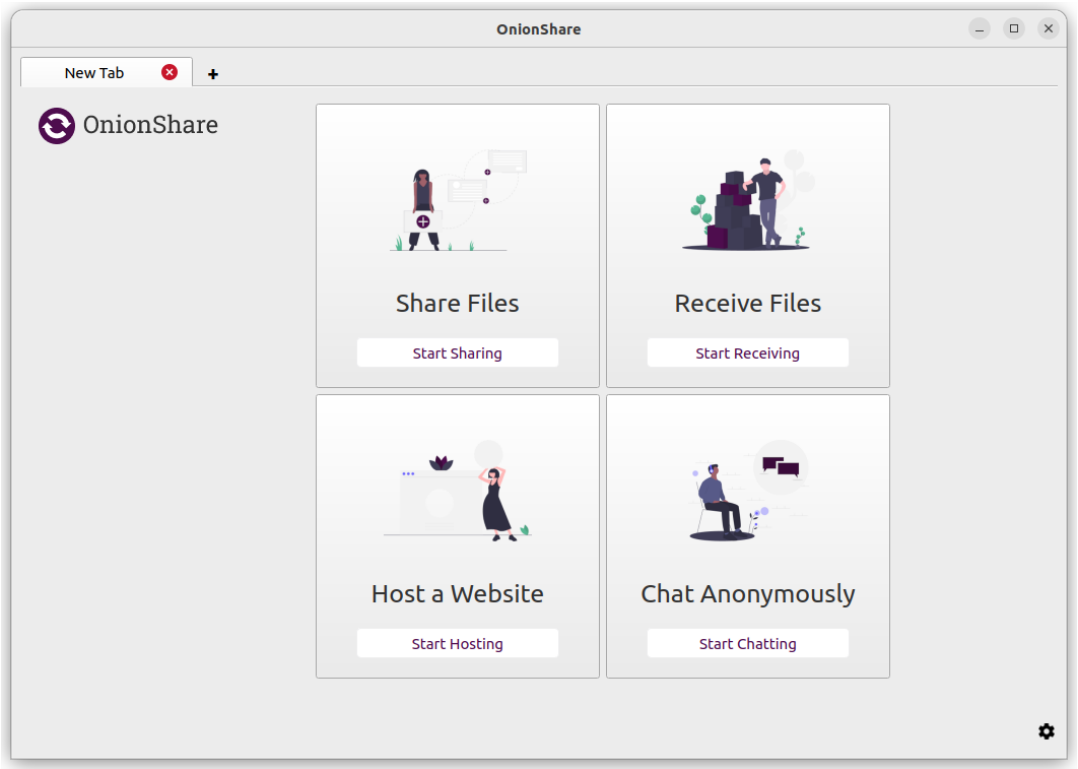
Tento nástroj je v době psaní vypnutý z důvodu nefunkčnosti s novými verzemi Toru. Aplikace se nachází na githubu a poslední datum aktualizace repozitáře je 5.12.2021[32]. Z tohoto důvodu se lze domnívat, že nástroj již aktualizován nejspíš nebude.



Obrázek 3.2: Upozornění na stránce *tor-relay.co*

3.2 OnionShare

Další možností je **OnionShare**. Tato aplikace slouží ke sdílení souborů, hostování onion služeb a chatování přes Tor. OnionShare je dostupný na operační systémy Windwos, MacOS, Linux, Android a iOS.

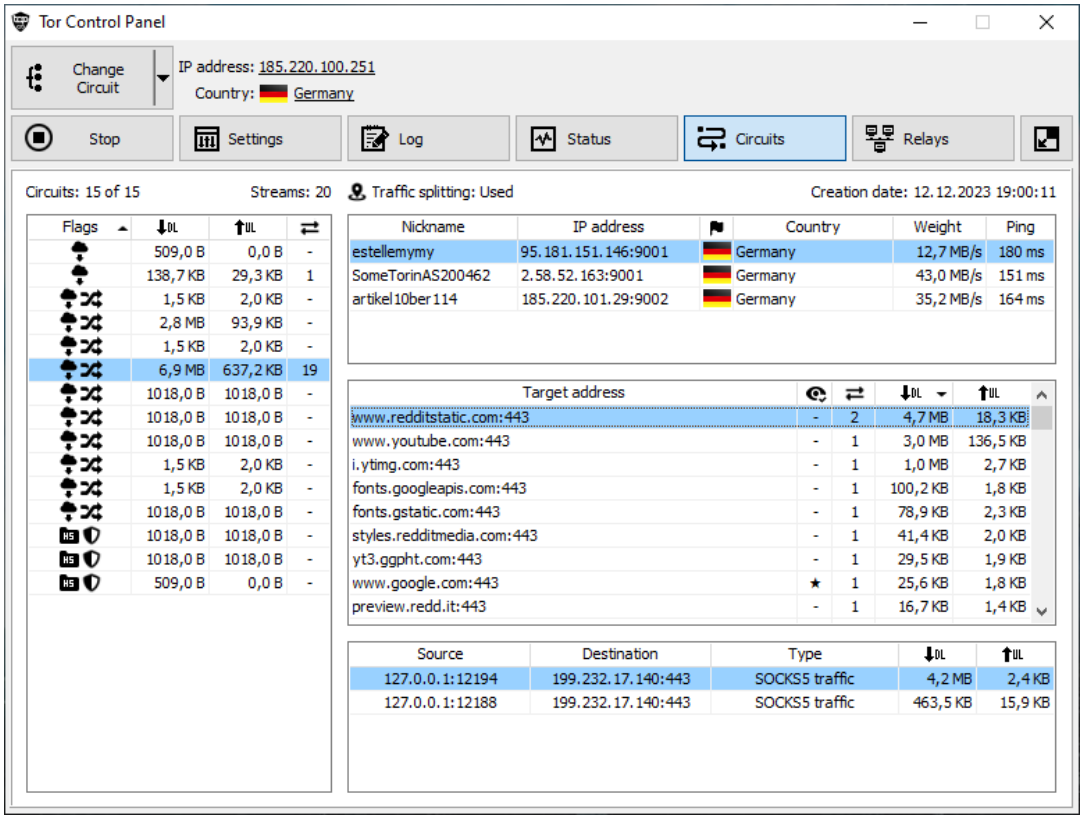


Obrázek 3.3: OnionShare[33]

3.3 Tor Control Panel

Poslední a nejobsáhlejší možností je **Tor Control Panel**. Jedná se o Windows aplikaci která nabízí možnost připojit se přes ni k síti Tor. Lze si zde nastavit preferované země a IP adresy pro sestavení Tor okruhu, je možné přes ni též skenovat uzly a zkontrolovat jejich dostupnost

či odezvu. Aplikace také umožňuje spravovat onion služby, non-exit i exit uzly a Obfs4 mosty. Aplikace je zaměřená spíše na uživatele, jenž chtějí mít větší kontrolu nad tím, jak jsou jejich data přes Tor posílána, jelikož jim k tomu poskytuje celou řadu funkcí.



Obrázek 3.4: Tor control panel [34]

Všechny tyto možnosti nabízí kvalitní služby pro práci se sítí Tor, ale náš projekt má pár specifických požadavků, kvůli kterým nám nevyhovuje ani jedna ze zmíněných aplikací. Prvním požadavkem je, aby aplikace běžela na Raspberry Pi. Kvůli tomu lze vyloučit aplikaci **Tor control panel**, jelikož podporuje pouze operační systém Windows, který dokáže být celkem hardwarově náročný a není vhodný pro námi vybraný počítač. Poté nám zbývá již pouze aplikace **OnionShare**, která ale nenaplňuje naše požadavky na možnost provozu non-exit uzlů a mostů. Dále by bylo vhodné mít možnost konfigurace služeb vzdáleně bez potřeby instalace dalšího softwaru na klientský PC pro přístup ke vzdálené ploše.

4. Metodika

Platforma se skládá ze tří částí. Hardware, který je tvořen Raspberry Pi počítačem, Serverová část běžící na Raspberry Pi a webová aplikace, která se spouští v prohlížeči u uživatele.

4.1 Server

Server je provozován na Raspberry Pi počítači na kterém běží operační systém **Raspberry Pi OS 12**, který je speciálně určen pro platformu Raspberry Pi. Serverová část aplikace je napsána v jazyce Java ve verzi 21. Jedná se o nejnovější LTS verzi tohoto jazyka.

Mezi hlaví úkoly serveru patří:

- Zpracování požadavků HTTP
- Server pracuje s konfiguračními torrc soubory Toru. Tyto soubory obsahují nastavení, která určují fungování jednotlivých služeb.
- Spouštění a zastavování Tor služeb.
- Sledování provozu na jednotlivých uzlech provozovaných na serveru a získávání informací o nich.

Pro vytvoření aplikace byl využit framework **Spring Boot**, který zjednodušuje tvorbu webových aplikací, obsahuje integraci s **Apache Tomcat** serverem a poskytuje framework pro vytváření RESTful API. Dále byla využita java knihovna **Lombok**, která automaticky generuje settery a gettery, čímž zvyšuje čitelnost kódu. Pro kompilaci byl použit **Maven**, který zároveň zjednodušuje správu závislostí aplikace

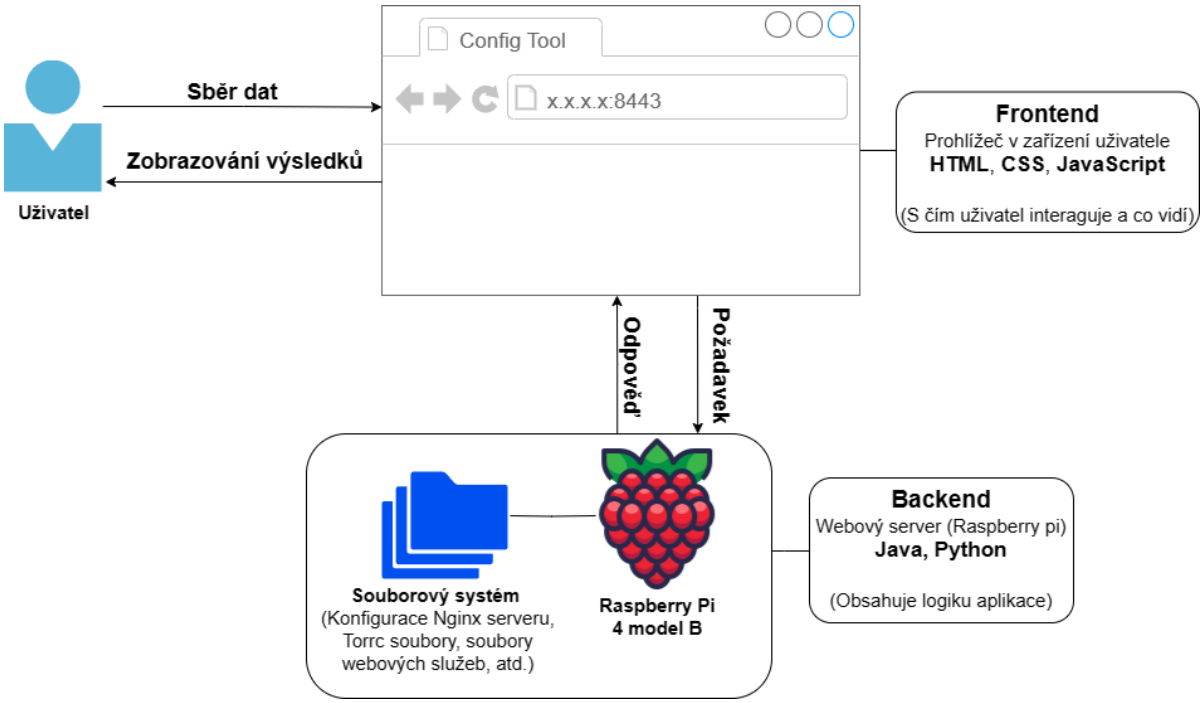
4.2 Webová aplikace

Webová aplikace poskytuje zjednodušený způsob správy a konfigurace služeb v síti Tor. Pomáhá uživateli snadno nastavit různé typy Tor uzlů (například non-exit uzly a mosty) a Onion služby. Aplikace umožňuje spouštět, zastavovat a upravovat konfigurace služeb a poskytuje

aktualizace stavu pro sledování jejich provozu. Kromě toho lze ovládat Tor Proxy, spravovat soubory relevantní pro nastavení Tor a vizualizovat provozní data procházející přes uzly. Webová aplikace je vytvořena pomocí **HTML, CSS, JavaScriptu, Bootstrapu** a **Thymelafu**.

4.3 Hardware

Platforma je postavena na **Raspberry Pi 4 Model B** ve verzi s 4 GB RAM.



Obrázek 4.1: Architektura webové aplikace

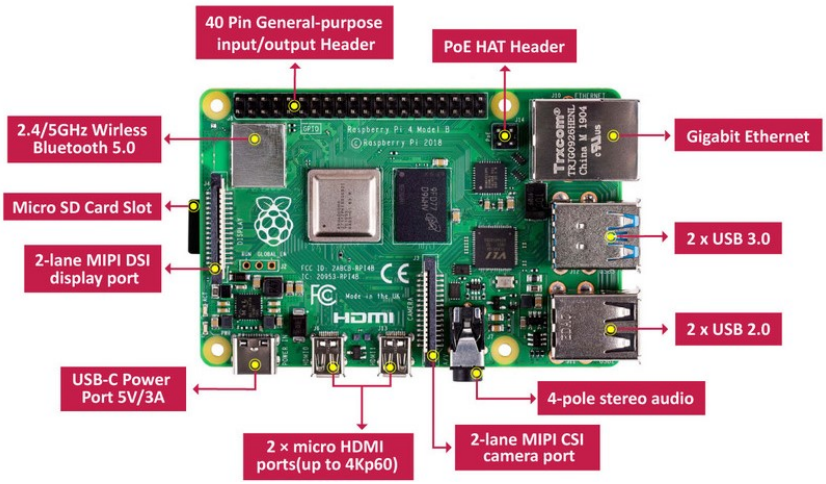
5. Výběr hardwaru

5.1 Raspberry Pi

Jako základ celé platformy byl zvolen počítač **Raspberry Pi** pro jeho nízkou cenu, spotřebu a kompaktní velikost. V době psaní této práce je již verze Raspberry Pi 5, ale při pořizování byla dostupná pouze verze 4. Hlavním parametrem výběru byla dostupná paměť. V následující tabulce je uveden přehled modelů a jejich dostupná operační paměť [16].

Model	Paměť (RAM)
Raspberry Pi 1 Model B	512 MB
Raspberry Pi 1 Model B+	512 MB
Raspberry Pi 2 Model B	1 GB
Raspberry Pi 3 Model A+	512 MB
Raspberry Pi 3 Model B	1 GB
Raspberry Pi 3 Model B+	1 GB
Raspberry Pi 4 Model B	1 GB, 2 GB, 4 GB, 8 GB
Raspberry Pi 5 Model B	4 GB, 8 GB

Dle požadavků na provoz uzlů je nejnáročnější výstupní uzel. Tomuto se ovšem v práci z dříve zmíněných důvodů nebudeme věnovat a proto se zaměříme na ostatní požadavky. Z námi používaných služeb je nejnáročnější tzv. "non-exit"uzel, pro který je doporučeno disponovat alespoň 512 MB RAM pro rychlosti do 40 Mbps a 1 GB RAM pro vyšší rychlosti [14]. Pro zajištění dostatečné paměťové kapacity pro spuštění potřebných služeb a konfigurační aplikace byl zvolen model Raspberry Pi 4 Model B s 4 GB RAM. Na obrázku níže je diagram popisující tento model. Z dostupných konektorů nám pro tuto práci budou stačit pouze USB-C pro napájení a microSD slot. Pro připojení k síti lze využít UTP kabel či Wi-fi. V této práci byla použita konektivita přes Wi-Fi.



Obrázek 5.1: Raspberry Pi 4 Model B [15]

5.2 Napájení

Dle oficiální stránky *RPishop.cz* se pro spolehlivou funkci Raspberry Pi 4 doporučuje 5V=3A zdroj [17]. V této práci byl pro napájení využit zdroj Xiaomi MDY-11-EZ, který nabízí dostatečný výkon [18].

5.3 Krabička

Počítač je umístěn v **Argon NEO Raspberry Pi 4** krabičce, která zároveň slouží jako pasivní chladič [19].

6. Výběr softwaru

6.1 Operační systém

Jako operační systém byl zvolen **Raspberry Pi OS** (Dříve znám jako Raspbian), jelikož se jedná o systém přímo určený pro Raspberry Pi. Raspberry Pi OS je postaven na linux distribuci Debian. Byla použita verze 12 (Bookworm) [20].

6.2 Programovací jazyky a technologie použité v aplikaci

6.2.1 Backend

Pro backend aplikace byl zvolen programovací jazyk **Java**. Hlavním důvodem byla autorova znalost tohoto jazyka. Při vývoji byla použita nejnovější LTS verze Java 21.

Pro specifickou část kódu byla použita knihovna napsaná v Pythonu. Tuto knihovnu a její využití si více přiblížíme v praktické části.

6.2.2 Frontend

Frontend webové aplikace využívá trojici základních webových technologií:

- **JavaScript**: Umožňuje interaktivitu a dynamické chování stránek. Validuje formuláře, načítá data dynamicky bez nutnosti znovu načíst stránku a komunikuje s backendem.
- **HTML**: Definuje strukturu a obsah stránek.
- **CSS**: Určuje vzhled a styl webových stránek.

6.2.3 Použité knihovny a framework

Spring Boot nabízí robustní základ pro vytváření aplikací v jazyce Java, ale může vyžadovat složité nastavení. Spring Boot tento proces zjednodušuje pomocí chytrých výchozích nastavení a automatické konfigurace, čímž značně omezuje množství šablonovitého kódu. Jako výchozí

vestavěný server používá Tomcat (Lze jednoduše přepnout na Jetty či Udertow) a umožňuje zabalit aplikaci jako samostatný soubor JAR.

Spring Security je framework navržený pro přidání zabezpečení do aplikací založených na platformě Spring. Řeší autentizaci (ověření, kdo je uživatel) a autorizaci (co smí dělat). Spring Security nabízí integrovanou ochranu proti běžným webovým útokům.

Thymeleaf je výkonný šablonovací engine jazyka Java známý svými přirozenými šablonami podobnými HTML, které podporují snadnou údržbu a použití. Bezproblémově se integruje se Spring MVC pro webové aplikace a nabízí flexibilitu při zpracování různých typů souborů (HTML, XML, JavaScript, CSS a plain text) [24].

Lombok je knihovna jazyka Java navržená tak, aby snížila množství šablonovitého kódu. Používá anotace k automatickému generování běžných prvků kódu, jako jsou gettery, settery a konstruktory.

Knihovna **WaifUPnP** zjednodušuje přesměrování portů UPnP, což je úloha, která je často potřebná pro komunikaci aplikací za routery s internetem. Umožňuje snadno otevírat a zavírat porty TCP i UDP a v aplikaci slouží k automatizaci této činnosti [25]. Pokud se v práci bavíme o portech v kontextu sítě Tor a jejích služeb, jedná se vždy o TCP porty, jelikož Tor podporuje pouze komunikaci přes TCP.

Stem je knihovna pro Tor v jazyce Python. Poskytuje řadu funkcí včetně možnosti komunikovat s control portem sítě Tor, interpretovat její deskriptory a využívat její skryté služby [26].

Bootstrap je front-end framework, který zjednodušuje tvorbu responzivních webových stránek. Nabízí knihovnu předpřipravených komponent HTML, CSS a JavaScriptu.

jQuery je JavaScript knihovna, která zjednodušuje vývoj webových aplikací, například manipulaci s prvky HTML, obsluhu událostí, vytváření animací a provádění požadavků AJAX.

Chart je JavaScript knihovna, která slouží k vytváření grafů.Podporuje různé typy grafů (čárové, sloupcové, koláčové atd.), které umožňují transformovat data do snadno pochopitelných vizuálních podob.

7. Praktická část

7.1 Instalace

Po instalaci operačního systému je nutné nainstalovat potřebný software. Pro přehlednost si instalaci rozdělíme do jednotlivých částí.

7.1.1 Jazyky a knihovny

Jako první nainstalujeme jazyk **Java 21**, který budeme potřebovat pro spuštění samotné webové aplikace. Poslední LTD verze Javy není v době psaní dostupná v repozitáři systému Raspberry Pi OS a je proto nutné tuto verzi nainstalovat jinak. K tomu lze využít nástroj **SDKMan**, který slouží k jednoduché instalaci různých Java verzí [27].

SDKMan nainstalujeme pomocí těchto příkazů:

```
$ curl -s "https://get.sdkman.io" | bash
$ source "$HOME/.sdkman/bin/sdkman-init.sh"
```

Tímto příkazem zobrazíme dostupné verze Javy:

```
$ sdk list java
```

```
================================================================================
Available Java Versions for Linux ARM 64bit
================================================================================
 Vendor        | Use | Version      | Dist    | Status     | Identifier
--------------------------------------------------------------------------------
 Corretto      |     | 21.0.2       | amzn    |            | 21.0.2-amzn
               |     | 21.0.1       | amzn    |            | 21.0.1-amzn
               |     | 17.0.10      | amzn    |            | 17.0.10-amzn
               |     | 17.0.9       | amzn    |            | 17.0.9-amzn
               |     | 11.0.22      | amzn    |            | 11.0.22-amzn
               |     | 11.0.21      | amzn    |            | 11.0.21-amzn
               |     | 8.0.402      | amzn    |            | 8.0.402-amzn
               |     | 8.0.392      | amzn    |            | 8.0.392-amzn
 Dragonwell    |     | 17.0.9       | albba   |            | 17.0.9-albba
               |     | 11.0.21      | albba   |            | 11.0.21-albba
               |     | 11.0.20      | albba   |            | 11.0.20-albba
```

Obrázek 7.1: Java verze dostupné přes SDKMan

Následně nainstalujeme zvolenou verzi.

```
$ sdk install java 21.0.2-open
```

Nyní potřebujeme již dříve zmíněnou Python knihovnu Stem. To se provede následujícím příkazem.

```
$ sudo apt install python3-stem
```

Python samotný instalovat nemusíme. Na Raspberry Pi OS je nainstalován defaultně.

7.1.2 Webový server

Pokud chce uživatel provozovat Onion službu či WebTunnel most, potřebuje k tomu webový server. Pro tento projekt byl zvolen **Nginx**.

```
$ sudo apt install nginx
```

7.1.3 Tor

Nyní potřebujeme nainstalovat Tor. Pro debian linux distribuce, kterou je i Raspberry Pi OS, se nedoporučuje využívat k instalaci Tor z debian repozitáře, jelikož se zde většinou nachází poslední LTS verze, která nemusí být vždy aktuální. Proto je doporučeno nainstalovat Tor z repozitáře balíčků Tor Projektu. Repozitář nabízí binární soubory pro architektury **amd64**, **arm64** a **i386**. Raspberry Pi využívá procesory architektury **arm64**. Pro práci s tímto repozitářem je třeba nainstalovat apt-transport-https.

```
$ sudo apt install apt-transport-https
```

V **/etc/apt/sources.list.d/** vytvoříme nový soubor s názvem *Tor.list*, do kterého přidáme následující dva řádky.

```
deb     [signed-by=/usr/share/keyrings/Tor-archive-keyring.gpg]
        https://deb.torproject.org/torproject.org
        <DISTRIBUTION> main
```

```
deb-src [signed-by=/usr/share/keyrings/Tor-archive-keyring.gpg]
        https://deb.torproject.org/torproject.org
        <DISTRIBUTION> main
```

<DISTRIBUTION> se nahradí kodovým označením operačního systému. V našem případě bookworm

Nyní podepíšeme balíčky.

```
$ sudo wget -qO- https://deb.torproject.org/torproject.org
    /A3C4F0F979CAA22CDBA8F512EE8CBC9E886DDD89.asc
    | gpg --dearmor | tee /usr/share/keyrings
    /Tor-archive-keyring.gpg >/dev/null
```

Nainstalujeme Tor a Tor debian keyring, abychom udržovali podpisový klíč aktuální.

```
$ sudo apt install
$ sudo apt install Tor deb.torproject.org-keyring
```

Dalším důležitým krokem je povolení automatických softwarových aktualizací, aby naše provozoavné Tor služby zůstali bezpečné.

```
$ sudo apt-get install unattended-upgrades apt-listchange
```

Do souboru **/etc/apt/apt.conf.d/50unattended-upgrades** přidáme následující řádky:

```
Unattended-Upgrade::Origins-Pattern {
    "origin=Debian,codename=${distro_codename},
    label=Debian-Security";
    "origin=TorProject";
};
Unattended-Upgrade::Package-Blacklist {
};
```

a do **/etc/apt/apt.conf.d/20auto-upgrades** přidáme tyto řádky:

```
APT::Periodic::Update-Package-Lists "1";
APT::Periodic::AutocleanInterval "5";
APT::Periodic::Unattended-Upgrade "1";
APT::Periodic::Verbose "1";
```

Tím jsme dokončili nastavování automatických aktualizací.

7.1.4 Mosty

Pro provoz mostů jsou potřeba balíčky **obfs4proxy** pro provoz Obfs4 mostu, **snowflake-proxy** pro provoz Snowflake proxy mostu.

```
$ sudo apt install -t bullseye-backports obfs4proxy
$ sudo apt install snowflake-proxy
```

Pro používání **WebTunnel** mostu je potřeba získat certifikát, který bude tento most využívat. Pro tuto potřebu je v našem projektu použit **acme.sh**. Jedná se o unix shell script implementující **acme client protokol**. Tento protokol slouží k automatizaci interakcí mezi certifikačními autoritami a servery uživatelů [28]. Dále pro kompilaci a běh WebTunnelu potřebujeme golang a následně naklonovat Webtunnel repozitář a sestavit jej.

```
$ curl https://get.acme.sh | sh -s email=<EMAIL>
$ apt install golang
$ git clone https://gitlab.torproject.org/tpo/
  anti-censorship/pluggable-transports/webtunnel
$ cd webtunnel/main/server
$ go build
$ sudo cp server /usr/local/bin/webtunnel
```

Pro správnou funkci WebTunnel je také třeba upravit AppArmor, který řídí přístup programů ke zdrojům. Do **/etc/apparmor.d/system_tor** vložíme následující řádek.

```
  /usr/local/bin/webtunnel ix,
```

Načteme upravený AppArmor profil.

```
$ sudo apparmor_parser -r /etc/apparmor.d/system_tor
```

Následuje tabulka pro lepší přehled toho, jaký software je potřeba při provozu jaké Tor služby.

Software	non-exit uzel	Onion služba	Obfs4	Snowflake	WebTunnel
Tor	✓	✓	✓	✓	✓
Nginx		✓			✓
obfs4proxy			✓		
snowflake-proxy				✓	
acme.sh					✓
golang					✓
webtunnel					✓

Pro zjednodušení instalace byl vytvořen **install.sh** skript, který instalaci a nastavení veškerého zmíněného softwaru automatizuje. Skript musí být spuštěn se sudo právy. Uživatel si zde vybere, jaký typ služeb si přeje provozovat a dle toho se nainstaluje potřebný software a provedou se potřebná nastavení. Následně je vytvořen *config.txt*, kde se uloží údaje o tom, jaké funkce uživatel při instalaci zvolil a dle toho se zpřístupní v aplikaci jednotlivé možnosti.

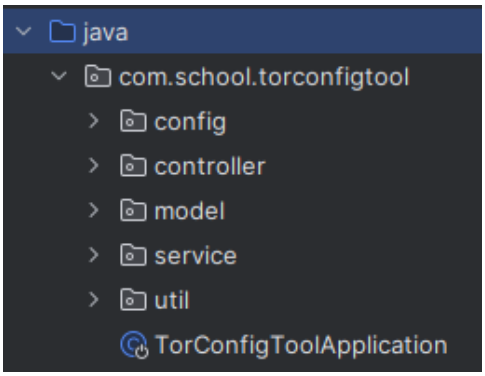
7.1.5 Vytvoření přihlašovacích údajů

I když je aplikace navržena s myšlenkou toho, že poběží pouze v lokální síti, považoval jsem i tak za vhodné zabezpečit přístup k této aplikaci. Pro tento účel byl využit již dříve zmíněný **Spring security framework**. Na konci instalace si uživatel vybere přihlašovací údaje (jméno a heslo), které jsou v programu **createAccount.jar** následně zahešovány pomocí metody BCryptPasswordEncoder, která používá BCrypt hašovací funkci. Poté údaje uloží do souboru *user* a ten uzamkne tak, aby nešel upravovat či smazat.

```
String username = args[0];
String password = args[1];
BCryptPasswordEncoder Encoder = new BCryptPasswordEncoder();
String hashedPassword = Encoder.encode(password);
String hashedUsername = Encoder.encode(username);
String userFileContent = hashedUsername + ":" + hashedPassword;
```

Obrázek 7.2: Hešování přihlašovacích údajů

7.2 Struktura aplikace

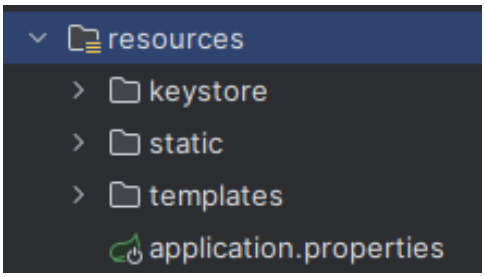


Obrázek 7.3: Balíčky aplikace

Serverová část aplikace je rozdělena do pěti balíčků viz obrázek 7.3.

- **config** balíček obsahuje třídu **WebSecurityConfig**, která je zodpovědná za konfiguraci zabezpečení aplikace, včetně kódování hesel, nastavení zabezpečení HTTP a ověřování uživatele.
- **controller** balíček obsahuje třídy, které mají buď to anotaci **@Controller** nebo **@RestController**. Třídy s anotací @Controller se starají o požadavky, které renderují pohledy pro uživatele. @RestController třídy vystavují endpointy, které se starají o zpracování požadavků přicházející na tyto endpointy.
- **model** balíček obsahuje třídy, které vytváří datovou strukturu a metody pro práci s nimi. Každá třída obsahuje anotaci **@Data**, která je součástí **Lombok** knihovny. Tato anotace automaticky vytvoří settery a gettery, čímž se zmenšuje objem potřebného kódu.

- Balíček **service** se skládá ze tříd, které obsahují byznys logiku aplikace. Tyto třídy mají anotaci **@Service**.
- V balíčku **util** se nachází pouze třída *Constants*, která slouží k deklaraci konstant použitých v aplikaci.



Obrázek 7.4: Struktura webové části aplikace

Strukturu webové části lze vidět na obrázku 7.4.

- Ve složce **keystore** se nachází self-signed certifikát a privátní klíč ve formátu .p12, aby se umožnila komunikace přes HTTPS. Hlavním důvodem pro tento ceritifikát je zabezpečení údajů při přihlašování se do aplikace.
- **Static** složka obsahuje složky **css**, ve které se nachází kaskádové styly a **js**, ve které se nachází javascript soubory.
- Ve složce **templates** se nachází *.html* soubory.
- Soubor *application.properties* obsahuje informace k nastavení SSL pro server.

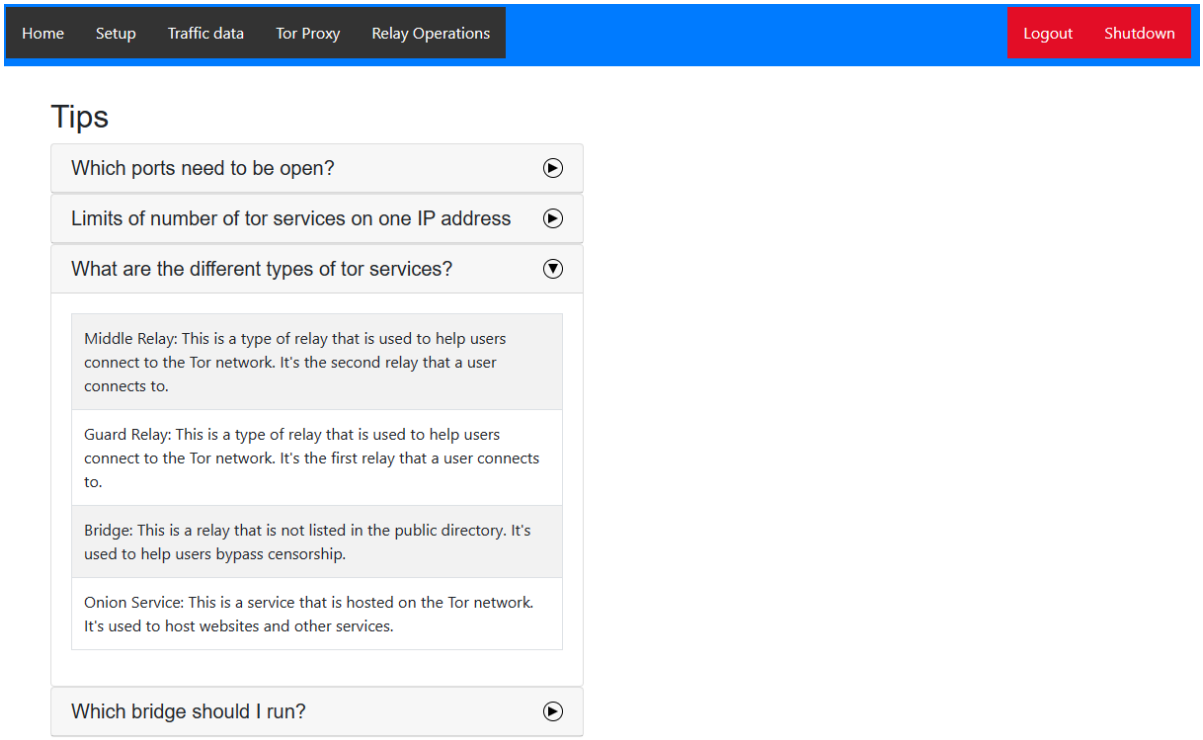
7.3 Funkcionalita aplikace

7.3.1 Autentizace

Při spuštění aplikace se musí uživatel nejdříve přihlásit. To probíhá na /login endpointu. Uživatel zde vyplní přihlašovací údaje, které jsou zahešovány stejnou funkcí jako při vytváření a následně porovnány s již uloženými údaji.

7.3.2 Domovská stránka

Po přihlášení se úživatel dostane na domovskou stránku. Zde se nachází pár tipů pro práci s Tor službami, jako jsou informace k tomu, jaké porty je třeba mít otevřené, limity Tor služeb, které lze provozovat na jedné IP adrese, druhy jednotlivých služeb a informace k tomu, jaký druh mostu by měl uživatel zvolit dle svých možností.

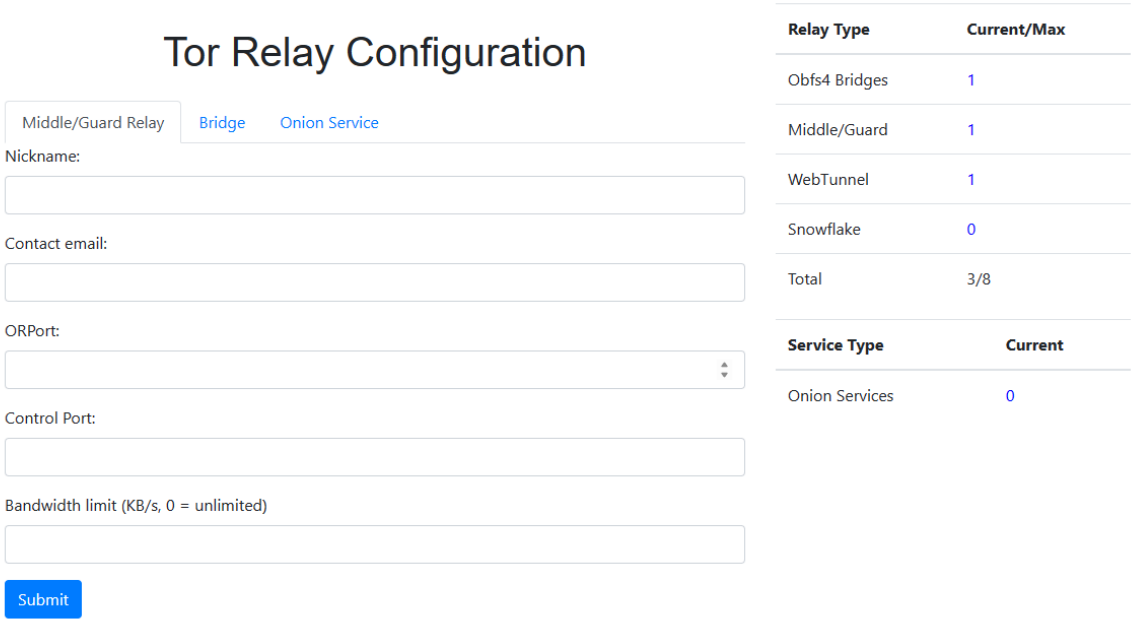


Obrázek 7.5: Domovská obrazovka

V levém horním rohu se nachází navigační menu, které slouží k navigaci po aplikaci. Toto menu se nachází na každé stránce v aplikaci. To samé platí o pravém horním rohu, kde se nachází dvě tlačítka. Jedno pro vypnutí aplikace a druhé pro odhlášení.

7.3.3 Vytváření nových služeb

Vytváření nových služeb probíhá na stránce *setup.html*.



Obrázek 7.6: Setup obrazovka

V levé části stránky je formulář, kde si uživatel volí mezi non-exit (Middle/Guard Relay) uzly, mosty (Bridge) a Onion služby (Onion Service). Do tohoto formuláře uživatel vyplní potřebné údaje pro každý typ služby.

Nastavení non-exit uzlu

Začneme non-exit uzly. Při nastavování non-exit uzlu je potřeba vyplnit následující údaje viz obrázek 7.6:

- **Nickname**: Přezdívka uzlu
- **Contact email**: Vaše emailová adresa na kterou můžete být kontaktováni v případě, že je váš uzel špatně nastaven či se s ním něco stane. Všechny emailové adresy jsou zveřejněny v deskriptorech spolu s vašimi uzly a jsou díky tomu indeovány například googlem, takže je lepší zvolit emailovou adresu, která není vaše hlavní, jelikož se na ni mohou zaměřit podvodníci.
- **ORPort**: Tento port bude inzerován pro příchozí připojení. Je potřeba, aby byl otevřený.
- **Control Port**: Tento port naslouchá lokálním připojením z Tor řídících aplikací. V našem případě je potřeba na sledování provozu.

- **Bandwidth limit**: Povolené množství provozu na uzlu.

Po vyplnění údajů je napřed ve formuláři zkontrolována jejich správnost. Mezi to patří například kontrola unikátnosti portů. Následně jsou údaje odeslány na endpoint */guard/configure*.

```
@PostMapping("/configure")
public String configureGuard(@RequestParam String relayNickname,
                             @RequestParam int relayPort,
                             @RequestParam String relayContact,
                             @RequestParam int controlPort,
                             @RequestParam Integer relayBandwidth,
                             Model model) {
    try {
        guardService.configureGuard(relayNickname, relayPort, relayContact, controlPort, relayBandwidth);
        model.addAttribute( attributeName: "successMessage",  attributeValue: "Tor Relay configured successfully!");
    } catch (Exception e) {
        model.addAttribute( attributeName: "errorMessage",  attributeValue: "Failed to configure Tor Relay: " + e.getMessage());
    }
    return "setup";
}
```

Obrázek 7.7: */guard/configure* endpoint

Tyto údaje jsou poté předány metodě *configureGuard* ve třídě **guardService**, kde proběhnou další kontroly. Zde se napřed pomocí metody *relayExists* zkontroluje, zda již neexistuje uzel se stejným názvem a metodou *portsAreAvailable* je zkontrolováno, zda nejsou zadané porty již použity u jiných Tor služeb. Všechny *torrc* konfigurační soubory jsou v naší aplikaci ukládány do složky **torrc**. Názvy *torrc* souborů jsou konstruovány následovně. *PREFIX-nickname_typ služby*. *PREFIX* je v našem případě *torrc*, aby bylo na první pohled poznat, že se jedná o *torrc* soubor.

```
public void configureGuard(String relayNickname, int relayPort, String relayContact, int controlPort, Integer relayBandwidth) throws Exception {
    String torrcFileName = TORRC_FILE_PREFIX + relayNickname + "_guard";
    Path torrcFilePath = Paths.get(TORRC_DIRECTORY_PATH, torrcFileName).toAbsolutePath().normalize();

    if (RelayUtilityService.relayExists(relayNickname)) {
        throw new Exception("A relay with the same nickname already exists.");
    }

    if (!RelayUtilityService.portsAreAvailable(relayNickname, relayPort, controlPort)) {
        throw new Exception("One or more ports are already in use.");
    }

    GuardConfig config = new GuardConfig();
    config.setNickname(relayNickname);
    config.setOrPort(String.valueOf(relayPort));
    config.setContact(relayContact);
    config.setControlPort(String.valueOf(controlPort));
    config.setBandwidthRate(String.valueOf(relayBandwidth));

    if (!torrcFilePath.toFile().exists()) {
        TorrcFileCreator.createTorrcFile(torrcFilePath.toString(), config);
    }
}
```

Obrázek 7.8: *configureGuard* metoda

Pokud vše proběhne v pořádku, je zavolána metoda *createTorrcFile* ze třídy **TorrcFileCreator**, která se stará o vytváření torrc souborů.

```
public static void createTorrcFile(String filePath, BaseRelayConfig config) {
    try (BufferedWriter writer = new BufferedWriter(new FileWriter(filePath))) {
        TorrcWriteConfigService torrcWriteConfigService = new TorrcWriteConfigService();
        torrcWriteConfigService.writeTorrcFileConfig(config, writer);
    } catch (IOException e) {
        throw new RuntimeException("Failed to create Torrc file", e);
    }
}
```

Obrázek 7.9: *createTorrcFile* metoda

K tomu využívá metodu *writeTorrcFileConfig*, která vytváří strukturu samotného torrc souboru a její obsah. Na konci této metody je zavolána ještě další metoda *writeSpecificConfig*. Tato metoda slouží k doplnění informací dle specificky nastavovaného druhu uzlu. V případě non-exit uzlů se zde žádná další nastavení nevyskytují. Tuto metodu si více popíšeme při nastavování mostů.

```
public void writeTorrcFileConfig(BaseRelayConfig config, BufferedWriter writer) throws IOException {
    writer.write( str: "Nickname " + config.getNickname());
    writer.newLine();
    String orPort = config.getOrPort() != null ? config.getOrPort() : "127.0.0.1:auto";
    writer.write( str: "ORPort " + orPort + " IPv4Only");
    writer.newLine();
    writer.write( str: "ContactInfo " + config.getContact());
    writer.newLine();
    writer.write( str: "ControlPort " + config.getControlPort());
    writer.newLine();
    writer.write( str: "CookieAuthentication 1");
    writer.newLine();
    writer.write( str: "SocksPort 0");
    writer.newLine();
    writer.write( str: "RunAsDaemon 1");
    writer.newLine();
    writer.write( str: "RelayBandwidthRate " + config.getBandwidthRate() + " KBytes");
    writer.newLine();

    String relayType = config.getClass().getSimpleName(); // Get the class name as the relay type
    String dataDirectoryPath = System.getProperty("user.dir") + File.separator + "torrc" + File.separator +
    writer.write( str: "DataDirectory " + dataDirectoryPath);
    writer.newLine();

    // Write specific configuration based on the type of relay
    writeSpecificConfig(config, writer);
}
```

Obrázek 7.10: *writeTorrcFileConfig* metoda

Po úspěšném nastavení našeho non-exit uzlu vznikne příslušný torrc soubor.

```
Nickname test
ORPort 9050 IPv4Only
ContactInfo test@email.com
ControlPort 9051
CookieAuthentication 1
SocksPort 0
RunAsDaemon 1
DataDirectory C:\Users\kouba\IdeaProjects\torConfigTool\torrc\dataDirectory\test_GuardConfig
RelayBandwidthRate 500 KBytes
```

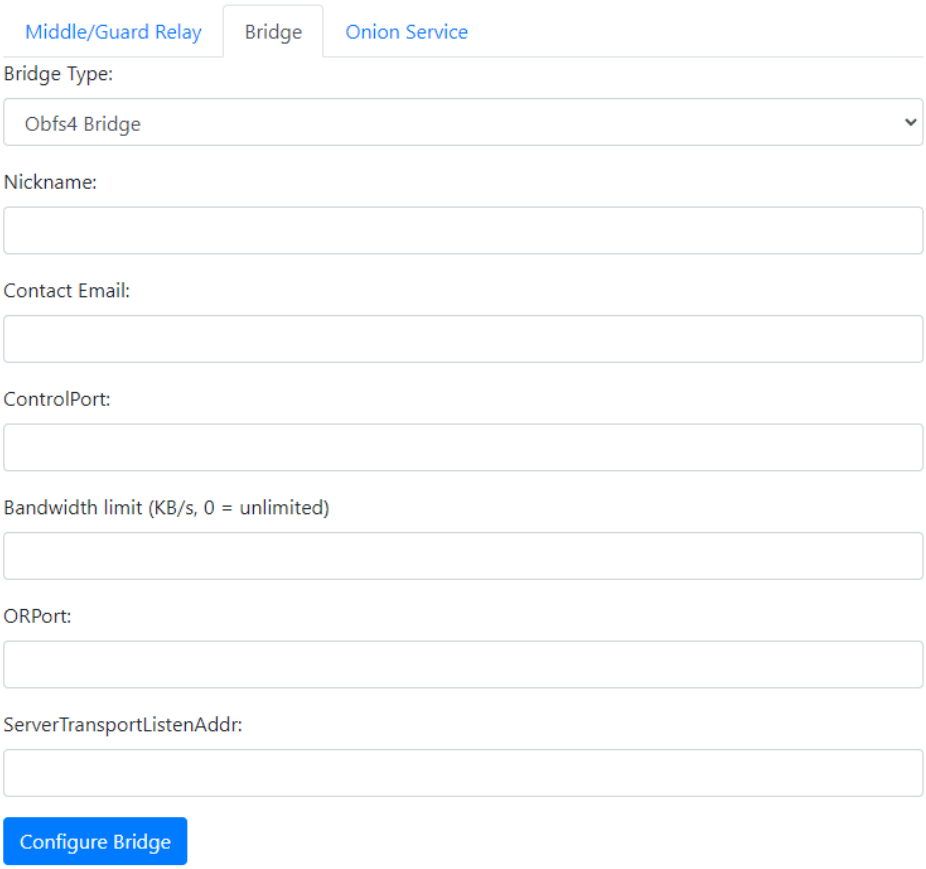
Obrázek 7.11: Torrc soubor non-exit uzlu

Jak lze vyčíst z obrázků 7.11 a 7.10, náš torrc soubor obsahuje více údajů, než jsme při nastavování zadávali. Tyto údaje navíc si teď vysvětlíme.

- **IPv4Only**: Toto značí, že náš uzel podporuje pouze IPv4. Tor má částečnou podporu pro IPv6, ale pro provoz služeb je stále vyžadována IPv4. Tato aplikace momentálně nemá podporu pro IPv6 a pracuje pouze s IPv4.
- **CookieAuthentication 1**: Pokud je tato možnost nastavena na 1, povolí se připojení na řídicím (control) portu pokud připojující se proces zná obsah souboru s názvem "*control_auth_cookie*", který Tor vytvoří ve svém datovém adresáři. Tento port je potřeba pro sledování provozu na uzlech.
- **SocksPort 0**: Tor defaultně otevírá SOCKS proxy na portu 9050. Nastavením SOCKSPort 0 říkáme, že tuto proxy nechceme, jelikož pro provoz uzlu není potřebná.
- **RunAsDaemon 1**: Proces poběží v pozadí.
- **DataDirectory**: Do této složky se ukládájí klíče, fingerprinty a další údaje o uzlech.

Nastavení mostu

Nastavení mostu probíhá do určité míry podobně, jako nastavení non-exit uzlu. Největší rozdíly jsou v tom, jak se zachází se zadanými údaji. Kvůli tomu, že existuje několik druhů mostu a každý z nich vyžaduje jiné údaje, je potřeba si s tím umět poradit.



Obrázek 7.12: Nastavení mostu

Po stisknutí *configure bridge* tlačítka je zavolán endpoint */bridge/configure*, který funguje obdobně jako */guard/configure* endpoint. Tento endpoint zpracovává samozřejmě jiné údaje. Některé údaje jsou označeny jako volitelné (required = false) z výše zmíněných důvodů.

```
@PostMapping("/configure")
public String configureBridge(@RequestParam String bridgeType,
                              @RequestParam(required = false) Integer bridgePort,
                              @RequestParam(required = false) Integer bridgeTransportListenAddr,
                              @RequestParam String bridgeContact,
                              @RequestParam String bridgeNickname,
                              @RequestParam(required = false) String webtunnelDomain,
                              @RequestParam int bridgeControlPort,
                              @RequestParam(required = false) String webtunnelUrl,
                              @RequestParam(required = false) Integer webtunnelPort,
                              @RequestParam(required = false) Integer bridgeBandwidth,
                              Model model) {
```

Obrázek 7.13: */bridge/configure* endpoint

Začneme u nastavení Obfs4 mostu. Jsou zde vyplňovány stejné údaje jako u již popisova-

ných non-exit uzlů. Jediným údajem navíc, který je třeba vyplnit, je **ServerTransportListenAddr** port. Tento port je pro naslouchání pluggable transport proxy, která funguje s daným mostem. V našem případě právě Obfs4.

Po vyplnění a odeslání údajů na server probíhá stejný proces, jako u non-exit uzlu. Po vytvoření Obfs4 mostu vznikne následující torrc soubor:

```
Nickname testBridge
ORPort 10002 IPv4Only
ContactInfo test@email.cz
ControlPort 10001
CookieAuthentication 1
SocksPort 0
RunAsDaemon 1
RelayBandwidthRate 0 KBytes
DataDirectory C:\Users\kouba\IdeaProjects\torConfigTool\torrc\dataDirectory\testBridge_BridgeConfig
BridgeRelay 1
ServerTransportPlugin obfs4 exec /usr/bin/obfs4proxy
ServerTransportListenAddr obfs4 0.0.0.0:10003
ExtORPort auto
```

Obrázek 7.14: Obfs4 Torrc soubor

Opět si vysvětlíme některé údaje, které se zde vyskytují navíc.

- **BrideRelay 1**: Nastaví uzel tak, aby fungoval jako most, pokud jde o předávání připojení od uživatelů mostu do sítě Tor. Způsobuje především to, že Tor publikuje deskriptor serveru do databáze mostu, která není veřejná.
- **ServerTransportPlugin obfs4**: Spustí pluggable transport Obfs4 nacházející se v */usr/-bin/obfs4proxy*.
- **ExtORPort auto**: Otevře tento port pro naslouchání připojení z pluggable transportu. Auto znamená, že se tento port nastaví automaticky při spuštění.

Nastavení Snowflake proxy a WebTunnel se už značně liší. Začneme tím jednodušším a tím je Snowflake proxy. V části, kde jsme mluvili o instalaci, jsme zmínili instalaci balíčku *snowflake-proxy*. Pro provoz Snowflake proxy stačí pouze nainstalovat tento balíček a pak pomocí příkazu *sudo systemctl start snowflake-proxy* tento balíček spustit. V našem případě to znamená, že při nastavování není třeba zadávat žádné údaje. Uživatel zde pouze klikne na

tlačítko **configure proxy**, čímž se pomocí metody *setupSnowflakeProxy* vytvoří soubor *Snowflake_proxy_configured*, který v aplikaci slouží pouze pro zaznamenání toho, že si uživatel přeje provozovat tento most.

```
public void setupSnowflakeProxy() {
    try {
        File snowflakeProxyRunningFile = new File(TORRC_DIRECTORY_PATH, child: "snowflake_proxy_configured");
        if (!snowflakeProxyRunningFile.createNewFile()) {
            throw new IOException("Failed to create file: " + snowflakeProxyRunningFile.getAbsolutePath());
        }

    } catch (IOException e) {
        throw new RuntimeException("Failed to create snowflake_proxy_configured file", e);
    }
}
```

Obrázek 7.15: Metoda *setupSnowflakeProxy*

Obrázek 7.16: Nastavení Snowflake proxy

Nyní přejdeme k nastavování WebTunnel mostu, které je o poznání složitější. Před konfigurací je napřed ověřit, že má uživatel otevřen porty 80 a 443. Ty jsou potřeba pro udělení certifikátu a jeho budoucí obnovování. Z tohoto důvodu jsou v aplikaci porty 80 a 443 rezervovány pro použití WebTunnel mostu. Pokud je povolen UPnP na routeru, není toto potřeba. Aplikace si potřebné porty otevře sama. První větší rozdíl oproti předchozím uzlům je ten, že je potřeba vlastnit doménu. Pro potřeby této práce byla zakoupena doména *tenmatys.eu* přes webhostingovou firmu **WEDOS**[29]. Zde je poté potřeba změnit DNS záznam na veřejnou IP adresu, kde bude WebTunnel provozován. Poté se opět vyplní formulář s potřebnými údaji. Oproti Obfs4 mostu zde není potřeba vyplňovat **ORPort**, ale musí být vyplněna příslušná **doména**, na které bude most provozován. Po odeslání formuláře probíhá opět stejný proces jako u předchozích až do poslední části, kde se musí nakonfigurovat web server, na kterém bude most provozován. To probíhá v následující části kódu uvnitř metody *configureBridge*

```java
if (webtunnelUrl != null && !webtunnelUrl.isEmpty()) {
    nginxService.generateIndexConfig();
    nginxService.changeRootDirectory(System.getProperty("user.dir") + "/onion/www/service-443");
    webtunnelService.setupWebtunnel(webtunnelUrl);
    String randomString = UUID.randomUUID().toString().replace( target: "-",  replacement: "").substring(0, 24);
    nginxService.modifyNginxDefaultConfig(System.getProperty("user.dir"), randomString, webtunnelUrl);
    config.setPath(randomString);
    webtunnelService.updateTorrcFile(config);

    nginxService.reloadNginx();
}
```

Obrázek 7.17: Kód nastavující webový server pro provoz WebTunnelu

Při konfiguraci serveru se napřed vytvoří defaultní index.html soubor ve složce */onion/www/service-443* pomocí metod *generateIndexConfig* a *createIndexHtmlFile*.

WebTunnel poběží na již dříve zmíněném Nginx webovém serveru. Při instalaci tohoto serveru byl vytvořen defaultní konfigurační soubor uvnitř */etc/nginx/sites-available/default*. Tuto konfiguraci lze použít k definování toho, jak by měl webový server zpracovávat příchozí požadavky. Nginx má dále složku pro "dostupné konfigurace"(*sites-available*) a složku pro "povolené konfigurace"(*sites-enabled*). To usnadňuje správu více virtuálních hostů (webových stránek) na jednom serveru.

První musíme v *default* souboru upravit cestu k našemu kořenovému adresáři, kde se nachází *index.html*. To obstará metoda *changeRootDirectory*. Server je třeba zabezpečit SSL certifikátem. Oficiální návod Tor projektu pro provozování WebTunnelu doporučuje dříve zmíněný acme.sh, který umožňuje získat certifikát od certifikačních autorit zdarma [31]. Získání certifikátu pomocí *acme.sh* je rozděleno do dvou částí. V první se generuje certifikát tímto příkazem:

```
$ acme.sh --issue -d example.com -w /home/wwwroot/example.com
  --nginx --server letsencrypt
```

Kde *example.com* je doména, pro kterou si přejeme certifikát vygenerovat a */home/wwwroot/example.com* je kořenová složka webu, kde jsou umístěny soubory webu. *–nginx* značí, že naše doména běží na Nginx serveru a *–server letsencrypt* určuje, kterou certifikační autoritu použijeme. V našem případě se jedná o **Let's Encrypt**[35]. To v našem programu obstarává metoda *generateCertificate*.

```java
public void generateCertificate(String webTunnelUrl, String programLocation) throws Exception {
    // Create the directory for the certificate files
    String certDirectory = programLocation + "/onion/certs/service-443/";
    File dir = new File(certDirectory);
    boolean isDirectoryCreated = dir.mkdirs();
    if (!isDirectoryCreated && !dir.exists()) {
        throw new IOException("Failed to create directory " + certDirectory);
    }

    // Generate the certificate
    String username = System.getProperty("user.name");
    String command = "/home/" + username + "/.acme.sh/acme.sh --issue -d " + webTunnelUrl + " -w " + programLocation
            + "/onion/www/service-443/ --nginx --server letsencrypt";

    Process certProcess = commandService.executeCommand(command);
    if (certProcess == null || certProcess.exitValue() != 0) {
        throw new Exception("Failed to generate certificate");
    }
}
```

Obrázek 7.18: metoda generateCertificate

Druhou částí získání ceritikátu je jeho instalace. To se provede nakopírováním vytvořených certifikačních souborů ze složky */.acme.sh/* do námi určené složky, odkud si server dané soubory načte. To se provede následujícím příkazem:

```
acme.sh --install-cert -d example.com \
--key-file       /path/to/keyfile/in/nginx/key.pem  \
--fullchain-file /path/to/fullchain/nginx/cert.pem \
--reloadcmd     "service nginx force-reload"
```

O toto se stará metoda *installCert*, která předá potřebné parametry (doménu, přezdívku uživatele systému, lokaci programu) metodě *createCertInstallationProcessBuilder*, která sestaví příkaz pro instalaci certifikátu. Následně je tento příkaz spuštěn.

```
private static ProcessBuilder createCertInstallationProcessBuilder(String webTunnelUrl, String username,
                                                                    String programLocation) {
    String command = "/home/" + username + "/.acme.sh/acme.sh --install-cert -d " + webTunnelUrl + " -d "
            + webTunnelUrl +
            " --key-file " + programLocation + "/onion/certs/service-443/key.pem" +
            " --fullchain-file " + programLocation + "/onion/certs/service-443/fullchain.pem" +
            " --reloadcmd";

    // Create a new process builder
    ProcessBuilder processBuilder = new ProcessBuilder();

    // Set the command for the process builder
    processBuilder.command("bash", "-c", command);
    return processBuilder;
}
```

Obrázek 7.19: metoda createCertInstallationProcessBuilder

Po instalaci certifikátu je třeba opět modifikovat konfigurační soubor. Tento soubor musíme upravit tak, aby server využíval nainstalovaný certifikát. Zároveň je třeba vytvořit tajnou cestu k našemu WebTunnelu, který se nachází na doméně. K tomu slouží příkaz *UUID.randomUUID().toString().r ", "").substring(0, 24);*, který vygeneruje string o délce 24 znaků. Tento string je poté použit jakožto naše tajná cesta. Dále je třeba nakonfigurovat reverzní proxy na našem serveru. Proxy server zachytí požadavky na tajnou cestu URL a přesměruje je na skrytý most Tor prostřednictvím WebTunnelu.

```
// Build the new configuration with port 443
String finalNginxConfig = String.format("""
    server {
        listen [::]:443 ssl http2;
        listen 443 ssl http2;
        root %s/onion/www/service-%d;
        index index.html index.htm index.nginx-debian.html;
        server_name $SERVER_ADDRESS;
        ssl_certificate %s/onion/certs/service-%d/fullchain.pem;
        ssl_certificate_key %s/onion/certs/service-%d/key.pem;
        location = /%s {
            proxy_pass http://127.0.0.1:%d;
            proxy_http_version 1.1;
            proxy_set_header Upgrade $http_upgrade;
            proxy_set_header Connection "upgrade";
            proxy_set_header Accept-Encoding "";
            proxy_set_header Host $host;
            proxy_set_header X-Real-IP $remote_addr;
            proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
            proxy_set_header X-Forwarded-Proto $scheme;
            add_header Front-End-Https on;
            proxy_redirect off;
            access_log off;
            error_log off;
        }
    }
    """, programLocation, webtunnelPort, programLocation, webtunnelPort, programLocation, webtunnelPort,
        randomString, transportListenAddr);
```

Obrázek 7.20: Konfigurace default Nginx konfiguračního souboru

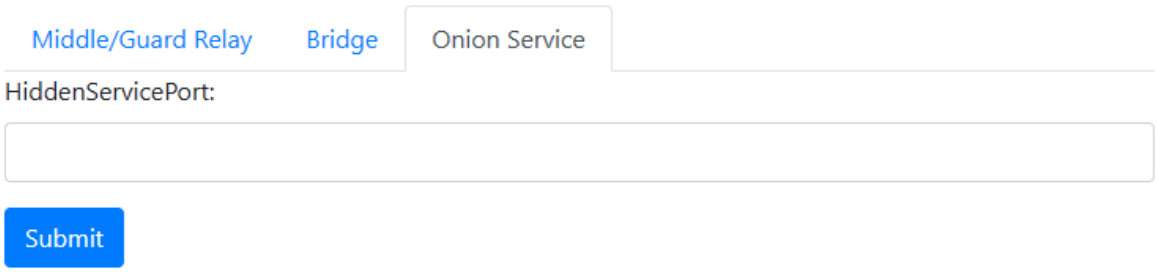
Nakonec se aktualizuje webTunnel torrc soubor s URL a tajnou cestou. Výsledný torrc soubor vypadá následovně.

```
1   Nickname testWebTunnel
2   ORPort 127.0.0.1:auto IPv4Only
3   ContactInfo test@email.com
4   ControlPort 9002
5   CookieAuthentication 1
6   SocksPort 0
7   RunAsDaemon 1
8   RelayBandwidthRate 0 KBytes
9   DataDirectory /home/matys/git/torConfigTool/torrc/dataDirectory/testWebTunnel_BridgeConfig
10  BridgeRelay 1
11  ServerTransportPlugin webtunnel exec /usr/local/bin/webtunnel
12  ServerTransportListenAddr webtunnel 127.0.0.1:15000
13  ServerTransportOptions webtunnel url=https://tenmatys.eu/663ae9481fc643deb65ee4f2
14  ExtORPort auto
```

Obrázek 7.21: WebTunnel torrc soubor

Nastavení onion služby

Při nastavování onion služby je potřeba vyplnit pouze to, na jakém portu tato služba poběží (**HiddenServicePort**).



Obrázek 7.22: Nastavení onion služby

Poté je zavolán */onion-service/configure* endpoint, který tento údaj zpracuje.

```
@PostMapping("/configure")
public String configureOnionService(@RequestParam int onionServicePort, Model model) {
    try {
        onionService.configureOnionService(onionServicePort);
        model.addAttribute( attributeName: "successMessage",  attributeValue: "Tor Onion Service configured successfully!");
    } catch (IOException e) {
        model.addAttribute( attributeName: "errorMessage",  attributeValue: "Failed to configure Tor Onion Service.");
    }
    return "setup";
}
```

Obrázek 7.23: */onion-service/configure endpoint*

Onion služba, stejně jako uzly, pracuje s *torrc* souborem. Proces vytvoření tohoto *torrc* souboru je stejný jako v předchozích případech. Onion služby běží také na našem Nginx serveru a proto je třeba udělat pár úprav zde. První je třeba vytvořit nový server block, což je konfigurační prvek v Nginx, který umožňuje hostovat více webových stránek na jednom serveru. Každý block má svou vlastní konfiguraci. Tato konfigurace zahrnuje informace o portu, na kterém má server poslouchat a kde se nachází složka se soubory serveru. O vytvoření server blocku se stará metoda *buildNginxServerBlock*. Konfigurace je uložena do složky *etc/nginx/sites-available*.

```
private String buildNginxServerBlock(int onionServicePort) {

    String currentDirectory = System.getProperty("user.dir");
    // Build the server block
    return String.format("""
            server {
                listen 127.0.0.1:%d;
                access_log /var/log/nginx/my-website.log;
                index index.html;
                root %s/onion/www/service-%d;
            }
            """, onionServicePort, currentDirectory, onionServicePort);
}
```

Obrázek 7.24: *buildNginxServerBlock* metoda

Dále je potřeba tuto novou konfiguraci nasadit. To se udělá vytvořením symbolického odkazu. Symbolický odkaz je speciální typ souboru, který odkazuje na jiný soubor nebo adresář. Funguje podobně jako zástupce na ploše. Když se pokusíte otevřít symbolický odkaz, operační systém vás přesměruje na skutečný soubor či adresář, na který odkaz ukazuje. Tento symbolický odkaz se vytvoří ve složce *etc/nginx/sites-enabled*, čímž se konfigurace aktivuje. O to se stará metoda *deployOnionServiceNginxConfig*

```
private void deployOnionServiceNginxConfig(String nginxConfig, int onionServicePort) {
    try {
        // Write the nginxConfig to a temporary file
        File tempFile = File.createTempFile( prefix: "nginx_config", suffix: null);
        try (BufferedWriter writer = new BufferedWriter(new FileWriter(tempFile))) {
            writer.write(nginxConfig);
        }

        String onionServiceConfigPath = "/etc/nginx/sites-available/onion-service-" + onionServicePort;

        // Use sudo to copy the temporary file to the actual nginx configuration file
        ProcessBuilder processBuilder = new ProcessBuilder( ...command: "sudo", "cp", tempFile.getAbsolutePath(),
                onionServiceConfigPath);
        Process process = processBuilder.start();
        process.waitFor();

        // Create a symbolic link to the nginx configuration file
        String enableConfigPath = "/etc/nginx/sites-enabled/onion-service-" + onionServicePort;
        processBuilder = new ProcessBuilder( ...command: "sudo", "ln", "-s", onionServiceConfigPath, enableConfigPath);
        process = processBuilder.start();
        process.waitFor();
```

Obrázek 7.25: *deployOnionServiceNginxConfig* metoda

Výsledný torrc soubor služby onion vypadá následovně.

```
HiddenServiceDir /home/matys/git/torConfigTool/onion/hiddenServiceDirs/onion-service-7850/
HiddenServicePort 80 127.0.0.1:7850
RunAsDaemon 1
SocksPort 0
DataDirectory /home/matys/git/torConfigTool/torrc/dataDirectory/onion_7850
```

Obrázek 7.26: torrc soubor služby onion

HiddenServiceDir určí, kam se budou ukládát informace o službě jako je hostname (doména onion služby).

7.3.4 Limitování Tor služeb

Tor síť má nastavené limity, kolik služeb je možno provozovat na jedné IP adrese. Tyto omezení Tor zavedl, aby zabrnáil takzvaným sybil útkoům. Sybil útok je typ útoku, při kterém útočník vytváří a ovládá velké množství uzlů (nebo identit) v peer-to-peer (P2P) síti, aby získal nepřiměřeně velký vliv nad touto sítí. Limit provoz jak non-exit uzlů tak mostů je osm na jedné IPv4 adrese[36]. Tato omezení jsou totožně nastavena i v aplikaci, kdy může uživatel vytvořit pouze 8 uzlů/mostů. O hlídání počtu jednotlivých uzlů/mostů se starají metody *getGuardCount* a *getBridgeCount*. *GetGuardCount* metoda projede *Torrc* složku a započítá soubory, jejichž jméno končí na *_guard*, jelikož tento sufix značí, že se jedná o torrc soubor non-exit uzlu.

```
public int getGuardCount() {
    File torrcDirectory = new File(TORRC_DIRECTORY_PATH);
    if (!torrcDirectory.exists() || !torrcDirectory.isDirectory()) {
        throw new RuntimeException("Directory " + TORRC_DIRECTORY_PATH + " does not exist or is not a directory.");
    }

    Matys
    String[] files = torrcDirectory.list(new FilenameFilter() {
        1 usage
        private static final String TORRC_FILE_SUFFIX = "_guard";

        Matys
        @Override
        public boolean accept(File dir, String name) {
            return name.startsWith(TORRC_FILE_PREFIX) && name.endsWith(TORRC_FILE_SUFFIX);
        }
    });

    return files != null ? files.length : 0;
}
```

Obrázek 7.27: getGuardCount metoda

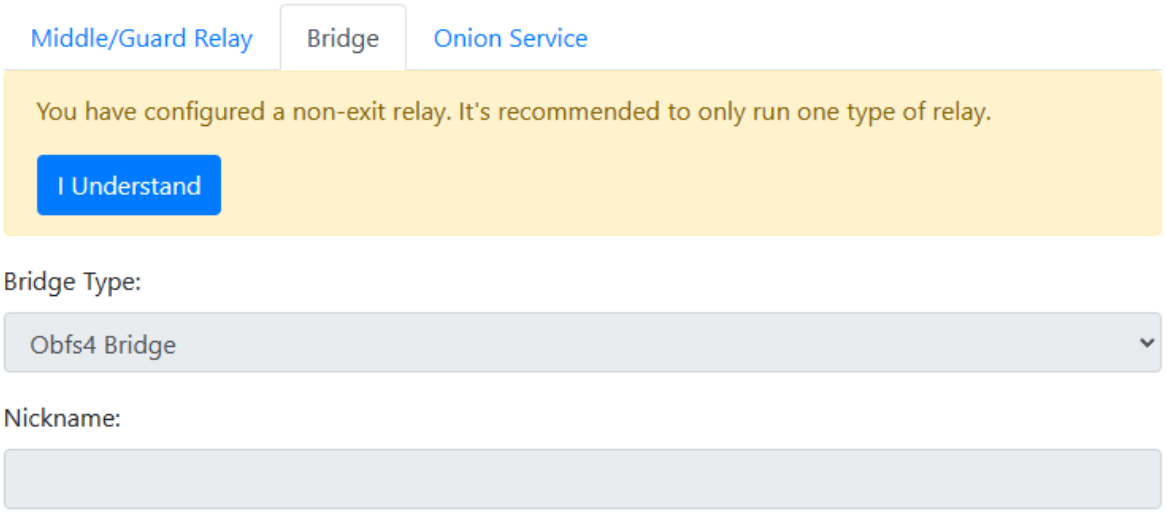
Metoda *getBridgeCount* funguje na podobném principu, ale má ještě pomocnou metodu

getConfiguredBridgeType, která projede torrc soubory a dle obsahu určí, o který typ mostu se jedná.

```
public Map<String, String> getConfiguredBridgeType() {
    File torrcDirectory = new File(TORRC_DIRECTORY_PATH);
    File[] files = torrcDirectory.listFiles((dir, name) -> name.startsWith(TORRC_FILE_PREFIX) &&
            name.endsWith("_bridge"));
    Map<String, String> runningBridgeTypes = new HashMap<>();

    if (files != null) {
        for (File file : files) {
            try {
                String content = new String(Files.readAllBytes(file.toPath()));
                String bridgeType = null;
                if (content.contains("obfs4")) {
                    bridgeType = "obfs4";
                } else if (content.contains("webtunnel")) {
                    bridgeType = "webtunnel";
                } else if (content.contains("snowflake")) {
                    bridgeType = "snowflake";
                }
                if (bridgeType != null) {
                    String bridgeNickname = file.getName().substring(TORRC_FILE_PREFIX.length(),
                            file.getName().length() - "_bridge".length());
                    runningBridgeTypes.put(bridgeNickname, bridgeType);
                }
            } catch (IOException e) {
                throw new RuntimeException("Error reading torrc file", e);
            }
        }
    }

    if (new File(torrcDirectory, child: "snowflake_proxy_configured").exists()) {
        runningBridgeTypes.put( k: "snowflake_proxy", v: "snowflake");
    }

    return runningBridgeTypes;
}
```

Obrázek 7.28: getGuardCount metoda

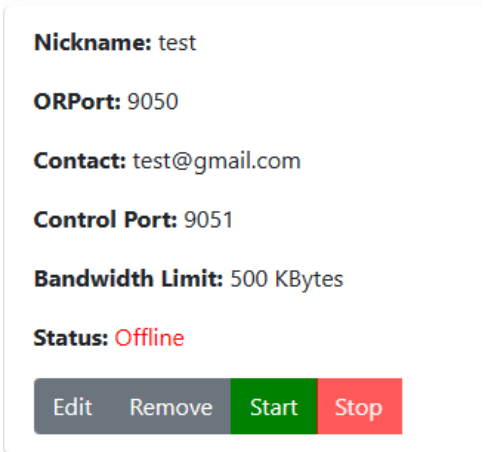
Další věc, kterou je doporučeno při provozu Tor služeb dodržovat, je provoz pouze mostu či uzlu na dané IP adrese v jeden moment. Například pokud uživatel provozuje non-exit užel, není doporučeno provozovat na stejné IP adrese most, jelikož je IP adresa uzlu veřejně zaznamenána a tím pádem je svázána s Torem, čímž se cenzura mostu stává o dost jednodušší a ten ztrácí svůj význam. V aplikaci na tohle není nastaveno přímo žádné omezení, aby měl uživatel volnost nastavení těch služeb, které si sám přeje. Pouze se uživateli zobrazí varování při konfiguraci nové služby.



Obrázek 7.29: Varování, které uživatele informuje, že již nakonfiguroval non-exit uzel

7.3.5 Přehled nakonfigurovaných služeb

Uživatel si může zobrazit jím nakonfigurované služby na stránce *relay-operations.html*. Zde se nachází přehled jednotlivých služeb a jejich vlastností. Pod každou službou se nachází tlačítka, kterými lze jednotlivé služby zapínat, vypínat, mazat či upravovat.



Obrázek 7.30: Nakonfigurovaný uzel

O přečtení torrc souborů, ze kterých se data zobrazují, se stará třída **TorConfigService**. V té se nachází metoda *readTorConfigurations*, která přečte všechny torrc soubory a zpracuje konfigurace na základě toho, o jaký typ služby se jedná. Se zpracováním pomáhá metoda *parseTorConfig*, která torrc soubory zpracuje do **TorConfig** objektu.

```
if (line.startsWith("Nickname")) {
    relayConfig.setNickname(line.split( regex: " ")[1].trim());
} else if (line.startsWith("ORPort")) {
    relayConfig.setOrPort(line.split( regex: " ")[1].trim());
} else if (line.startsWith("Contact")) {
    relayConfig.setContact(line.split( regex: " ")[1].trim());
} else if (line.startsWith("HiddenServiceDir")) {
    config.setHiddenServiceDir(line.split( regex: " ")[1].trim());
} else if (line.startsWith("HiddenServicePort")) {
    String[] parts = line.split( regex: " ");
    String addressAndPort = parts[parts.length - 1];
    String port = addressAndPort.split( regex: ":")[1];
    config.setHiddenServicePort(port);
} else if (line.startsWith("ControlPort")) {
    relayConfig.setControlPort(line.split( regex: " ")[1].trim());
} else if (line.startsWith("RelayBandwidthRate")) {
    String bandwidthRate = line.substring("RelayBandwidthRate".length()).trim();
    updateBandwidthRate(relayConfig, bandwidthRate);
} else if (line.startsWith("ServerTransportListenAddr obfs4") && relayType.equals("bridge")) {
    ((BridgeConfig) relayConfig).setServerTransport(line.substring(line.indexOf("obfs4")).trim());
} else if (line.startsWith("ServerTransportOptions webtunnel url") && relayType.equals("bridge")) {
    String fullUrl = line.split( regex: "=")[1].trim();
    try {
        java.net.URI uri = new java.net.URI(fullUrl);
        String webtunnelUrl = uri.getHost();
        String path = uri.getPath().substring( beginIndex: 1);
        ((BridgeConfig) relayConfig).setWebtunnelUrl(webtunnelUrl);
        ((BridgeConfig) relayConfig).setPath(path);
    } catch (URISyntaxException e) {
        throw new RuntimeException(e);
    }
}
```

Obrázek 7.31: Zpracování Torrc souborů v metodě *parseTorConfig*

Tyto TorConfigy jsou následně metodou *prepareModelForRelayOperationsView* přidány do modelu.

```
public String prepareModelForRelayOperationsView(Model model) {
    addRelayConfigsToModel(model);
    addHostnamesToModel(model);
    addUPnPPortsToModel(model);
    return "relay-operations";
}

1 usage  new *
private void addRelayConfigsToModel(Model model) {
    model.addAttribute( attributeName: "guardConfigs", torConfigService.readTorConfigurations
            (Constants.TORRC_DIRECTORY_PATH, expectedRelayType: "guard"));
    model.addAttribute( attributeName: "bridgeConfigs", torConfigService.readTorConfigurations
            (Constants.TORRC_DIRECTORY_PATH, expectedRelayType: "bridge"));
    model.addAttribute( attributeName: "onionConfigs", torConfigService.readTorConfigurations
            (Constants.TORRC_DIRECTORY_PATH, expectedRelayType: "onion"));
}
```

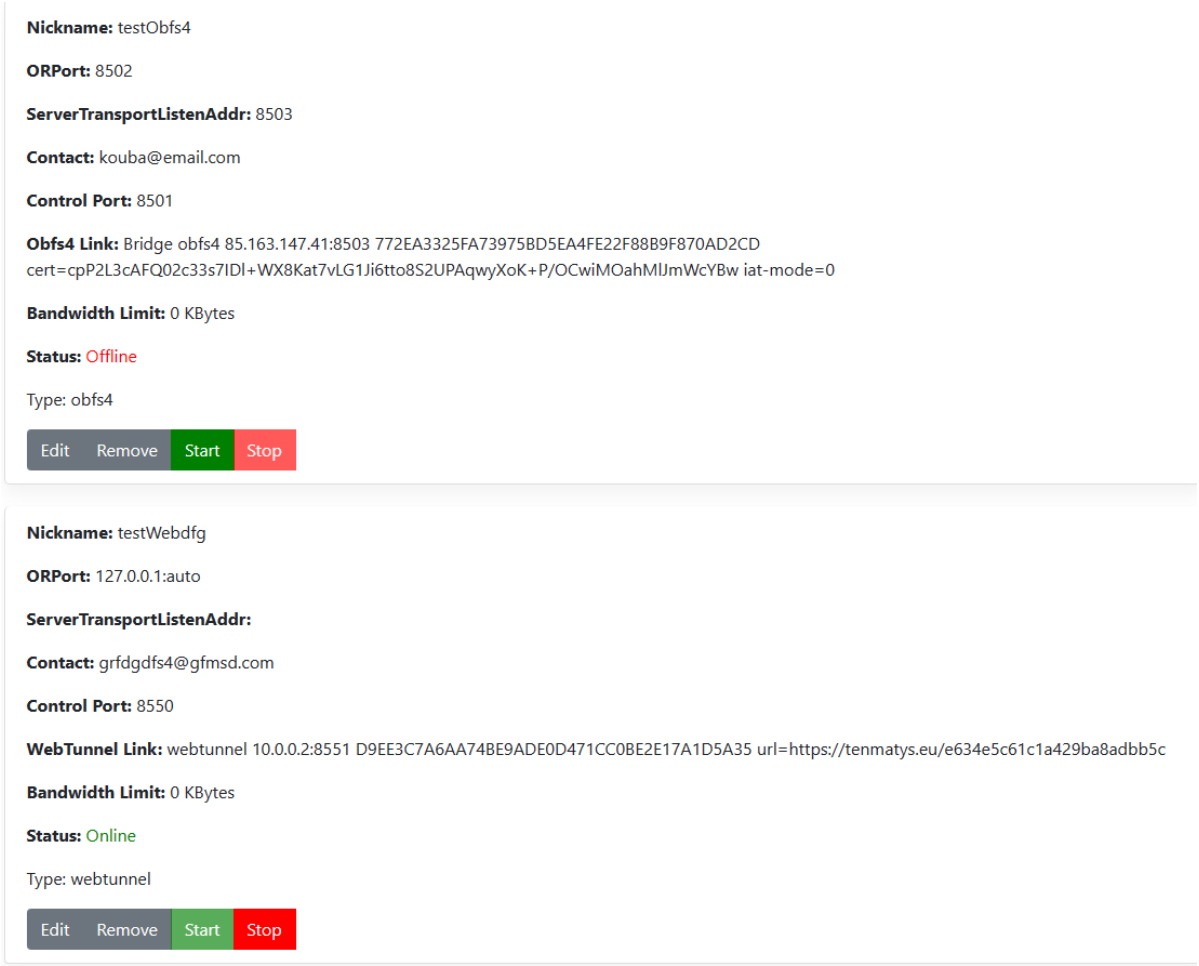
Obrázek 7.32: Přidání TorConfig objektů do modelu

Thymeleaf, který je použitý v **relay-operations.html**, přistoupí k modelům a načte si jednotlivé konfigurace, ze kterých následně vygeneruje HTML obsah stránky. Například řádek *<div th:each="config : $guardConfigs"class="list-group-item»* v **relay-operations.html** iteruje přes každý objekt **TorConfig** v seznamu **guardConfigs** z modelu. Pro každý **TorConfig** vygeneruje prvek div s třídou list-group-item a naplní jeho obsah konfiguračními daty.

```
<div th:each="config : ${guardConfigs}" class="list-group-item">
    <p><strong>Nickname:</strong> <span th:text="${config.getGuardConfig.nickname}"></span></p>
    <p><strong>ORPort:</strong> <span th:text="${config.getGuardConfig.orPort}"></span></p>
    <p><strong>Contact:</strong> <span th:text="${config.getGuardConfig.contact}"></span></p>
    <p><strong>Control Port:</strong> <span th:text="${config.getGuardConfig.controlPort}"></span></p>
    <p><strong>Bandwidth Limit:</strong> <span th:text="${config.getGuardConfig.bandwidthRate}"></span></p>
    <p><strong>Status:</strong> <span th:id="${'status-' + config.getGuardConfig.nickname}">Unknown</span></p>
```

Obrázek 7.33: Generování HTML pomocí Thymeleaf pro guard uzly

Stejným způsobem je vytvářen HTML obsah i pro všechny typy mostů. Zde se krom těchto metod nachází i pomocné metody *getWebtunnelLink* a *getObfs4Link*, které z informací z torrc souboru a dataDirectory jako je fingerprint, sestaví odkazy na tyto mosty. Odkazy pak lze použít buď to pro otestování funkčnosti mostu či poskytnutí někomu, kdo by chtěl daný most používat.



Obrázek 7.34: Nakonfigurované Obfs4 a WebTunnel mosty

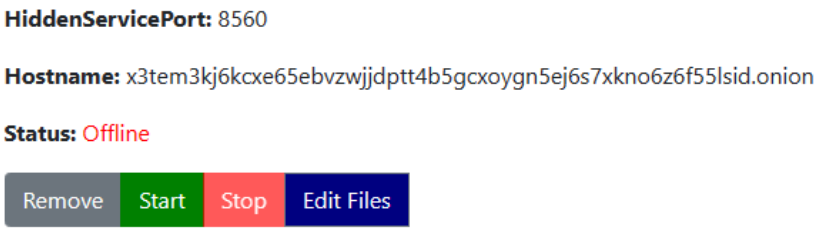
```java
public String getWebtunnelLink(String relayNickname) {
    String dataDirectoryPath = System.getProperty("user.dir") + File.separator + "torrc" + File.separator
            + "dataDirectory";
    String fingerprintFilePath = dataDirectoryPath + File.separator + relayNickname + "_BridgeConfig"
            + File.separator + "fingerprint";
    String fingerprint = torFileService.readFingerprint(fingerprintFilePath);

    String torrcFilePath = System.getProperty("user.dir") + File.separator + "torrc" + File.separator
            + TORRC_FILE_PREFIX + relayNickname + "_bridge";

    String webtunnelDomainAndPath = null;
    String webtunnelPort = null; // Changed from int to String
    try (BufferedReader reader = new BufferedReader(new FileReader(torrcFilePath))) {
        String line;
        while ((line = reader.readLine()) != null) {
            if (line.startsWith("ServerTransportOptions webtunnel url")) {
                webtunnelDomainAndPath = line.split( regex: "=")[1].trim();
            }
            if (line.startsWith("# webtunnel")) {
                webtunnelPort = line.split( regex: " ")[2];
            }
        }
    } catch (IOException e) {
        throw new RuntimeException("Failed to read torrc file", e);
    }

    return "webtunnel 10.0.0.2:" + webtunnelPort + " " + fingerprint + " url=" + webtunnelDomainAndPath;
}
```

Obrázek 7.35: metoda *getWebtunnelLink*

Při generování HTML obsahu pro Onion služby je použita metoda *readHostnameFile*, která nám do modelu přidá doménu naší onion služby. Tuto doménu se dočte z **HiddenServiceDir**, kterou jsme si popsali při konfiguraci Onion služby.

```java
public String readHostnameFile(int port) {
    // Get the current working directory
    String currentDirectory = System.getProperty("user.dir");

    // Build the correct path to the hostname file
    Path path = Paths.get(currentDirectory, ...more: "onion", "hiddenServiceDirs", "onion-service-"
            + port, "hostname");
    try {
        return new String(Files.readAllBytes(path));
    } catch (IOException e) {
        return "Unable to read hostname file";
    }
}
```

Obrázek 7.36: metoda *readHostnameFile*



Obrázek 7.37: Nakonfigurovaná Onion služba

7.3.6 Zapínání/vypínání služeb

Ve *front-endu* zpracovává soubor JavaScript start.js události kliknutí na tlačítka start a stop.js události kliknutí na tlačítka stop. Při kkliknutí na jedno z těchto tlačítek se z atributů načte přezdívka a typ služby. Při zmáčknutí se zobrazí spinner, který signalizuje, že proces spouštění či zastavování probíhá.

```
$.ajax({
    url: '/relay-operations-api/start',
    type: 'POST',
    data: {
        relayNickname: nickname,
        relayType: relayType
    },
    success: function (data) : void  {
        console.log('Relay started:', data);
        // Call the updateRelayStatus function
        updateRelayStatus(nickname, relayType);
    },
    error: function (error) : void  {
        console.error('Error starting relay:', error);
    },
    complete: function () : void  {
        // Enable the buttons and hide the spinner
        startButton.prop('disabled', false);
        stopButton.prop('disabled', false);
        editButton.prop('disabled', false);
        spinner.hide();
    }
});
```

Obrázek 7.38: Ajax požadavek na spuštění služby

Endpoint pro požadavek je buď */relay-operations-api/start*, nebo */relay-operations-api/stop* podle toho, zda bylo kliknuto na tlačítko start nebo stop. Kontrolér si poté zavolá metody startRelay respektive stopRelay ze třídy RelayOperationsService.

```
public String startRelay(String relayNickname, String relayType, Model model) {
    String view;
    if ("guard".equals(relayType)) {
        view = changeRelayState(relayNickname, relayType, model, start: true, updateFingerprint: true);
    } else {
        view = changeRelayState(relayNickname, relayType, model, start: true, updateFingerprint: false);
    }

    new Thread(() -> {
        try {
            relayStatusService.waitForStatusChange(relayNickname, relayType, expectedStatus: "online");
            upnpService.openOrPort(relayNickname, relayType);
        } catch (InterruptedException e) {
            throw new RuntimeException("Error while waiting for relay to start", e);
        }
    }).start();

    return view;
}
```

Obrázek 7.39: Metoda *startRelay*

Spuštění služeb probíhá pomocí příkazu *Tor -f* a specifikací cesty ke konfiguračnímu souboru. *-f* značí, že chceme specifikovat souboru

```
String command = "sudo tor -f " + torrcFilePath.toAbsolutePath();
try {
    commandService.executeCommand(command);
} catch (RuntimeException e) {
    throw new RuntimeException("Failed to start Tor Relay service.", e);
}
```

Obrázek 7.40: Příkaz pro spuštění služby

Při zastavování služby je nejdříve potřeba zjistit PID této služby. To se provede následujícím příkazem:

```
$ ps aux | grep -P '\\b%s\\b' | grep -v grep | awk '{print $2}'
```

Tento příkaz vyhledá ID procesu (PID) konkrétního spuštěného procesu. Nejprve získá kompletní seznam všech běžících procesů. Poté tento seznam vyfiltruje a ponechá pouze řádky, které obsahují přesně požadované slovo (toto slovo nahradí část '%s', která je v našem případě přezdívka naší Tor služby). Nakonec ze zbývajícího výstupu vyčlení druhý sloupec, ve kterém

se nachází ID procesu. Toto ID je následně použito k zastavení dané služby. V aplikaci se o toto stará metoda *getTorRelayPID*.

```
public int getTorRelayPID(String torrcFilePath) {
    String relayNickname = new File(torrcFilePath).getName();
    String command = String.format("ps aux | grep -P '\\b%s\\b' | grep -v grep | awk '{print $2}'", relayNickname);

    try {
        List<String> outputLines = CommandService.getCommandOutput(command);

        // Assuming the PID is on the first line, if not, you need to check the outputLines list.
        if (!outputLines.isEmpty()) {
            String pidString = outputLines.getFirst();
            return Integer.parseInt(pidString);
        } else {
            return -1;
        }
    } catch (IOException e) {
        return -1;
    }
}
```

Obrázek 7.41: Metoda *getTorRelayPID*

K zastavení služby je použit příkaz *kill -SIGINT + pid*. Vypínání probíhá tak, že služba přestane přijímat nová spojení a po třiceti sekundách se vypne. Po opětovném zmáčknutí stop tlačítka lze toto vypnutí urychlit, ale připojeným uživatelům se tímto může přerušit spojení.

7.3.7 MyFamily atribut

Tor vyžaduje, aby provozovatelé guard, middle a exit uzlů vyplňovaly *MyFamily* atribut, který se zapisuje do torrc souboru. Tento atribut by měl zahrnovat fingerprinty všech uzlů provozovaných operátorem (nikoliv mostů), z toho důvodu, aby Tor při vytváření řetězce nepoužil více uzlů od jednoho operátora, čímž by mohlo dojít k ohrožení anonymity uživatele sítě. Fingerprint uzlu se nachází v jeho dataDirectory složce. O tento atribut se stará metoda *getGuardRelayFingerprints*, která projede dataDirectory složky nakonfigurovaných uzlů a získá jejich fingerprinty. Metoda *updateTorrcWithFingerprints* následně tyto fingerprinty vypíš do torrc souborů. Tento atribut je potřeba vyplnit pro všechny uzly.

```
private List<String> getGuardRelayFingerprints() {
    // This path lead to the base directory where all relay data directories are stored
    String dataDirectoryPath = System.getProperty("user.dir") + File.separator + "torrc" + File.separator
            + "dataDirectory";
    File dataDirectory = new File(dataDirectoryPath);
    File[] dataDirectoryFiles = dataDirectory.listFiles(File::isDirectory);

    List<String> fingerprints = new ArrayList<>();
    if (dataDirectoryFiles != null) {
        for (File dataDir : dataDirectoryFiles) {
            // Check if the directory name ends with "_guard" to ensure it's a guard relay
            if (dataDir.getName().endsWith("_GuardConfig")) {
                String fingerprintFilePath = dataDir.getAbsolutePath() + File.separator + "fingerprint";
                String fingerprint = torFileService.readFingerprint(fingerprintFilePath);
                if (fingerprint != null) {
                    fingerprints.add(fingerprint);
                }
            }
        }
    }
    return fingerprints;
}
```

Obrázek 7.42: Metoda *getGuardRelayFingerprints*

7.3.8 Získávání statusu služby

Na backendu má třída **RelayStatusService** metodu *getRelayStatus*. Tato metoda získá stav uzlu načtením PID (ID procesu). Tato metoda provede příkaz pro získání PID a vrátí jej. Pokud je PID větší než 0, je uzel považován za "online". Pokud je PID roven -1, je uzel považován za "offline".

```
public String getRelayStatus(String relayNickname, String relayType) {
    String torrcFilePath = torFileService.buildTorrcFilePath(relayNickname, relayType).toString();
    int pid = getTorRelayPID(torrcFilePath);
    return pid > 0 ? "online" : (pid == -1 ? "offline" : "error");
}
```

Obrázek 7.43: Metoda *getRelayStatus*

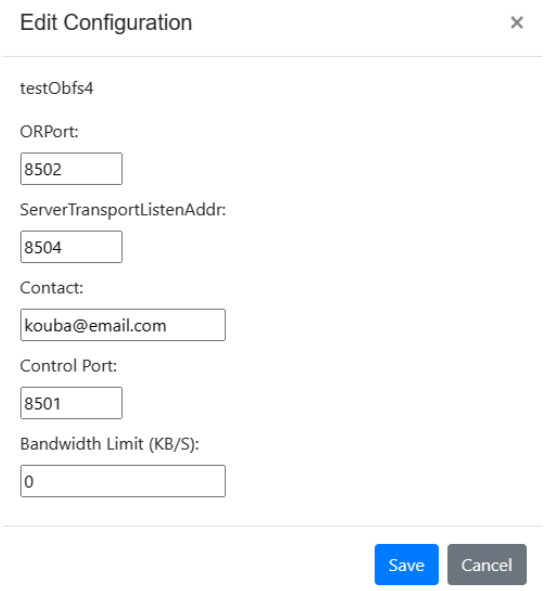
Na frontendu obsahuje status.js funkci updateRelayStatus. Tato funkce odešle požadavek AJAX GET na endpoint /relay-operations-api/status s přezdívkou uzlu a typem jako parametry. Server odpoví stavem uzlu, který se pak použije k aktualizaci stavu zobrazeného na webové stránce.Tato funkce je volána pro každý uzel při načtení stránky a poté pravidelně každých 5 sekund, aby byl stav aktualizován.

```
$(document).ready(function () {
    // Function to check and update relay status
    function updateRelayStatus(nickname, relayType) {
        $.get("/relay-operations-api/status?relayNickname=" + nickname + "&relayType=" + relayType, function (data) {
            const statusElement = $("#status-" + nickname);
            if (data === "online") {
                statusElement.text("Online");
                statusElement.css("color", "green");
            } else if (data === "offline") {
                statusElement.text("Offline");
                statusElement.css("color", "red");
            } else {
                statusElement.text("Unknown");
                statusElement.css("color", "blue");
            }
        });
    }

    // Function to update relay status for all relays
    function updateAllRelayStatus() {
        $(".start-button").each(function () {
            const nickname = $(this).data("config-nickname");
            const relayType = $(this).data("config-type");
            updateRelayStatus(nickname, relayType);
        });
    }
```

Obrázek 7.44: *status.js*

7.3.9 Editování služeb

U jednotlivých služeb lze editovat některé jejich atributy. To se provede stisknutím tlačítka *edit*. Po stisknutí se objeví editační okno.



Obrázek 7.45: Editační okno pro Obfs4 most

Napřed se zjistí, o který typ služby se jedná. Poté se načte aktuální konfigurace do editačního formuláře. O to se stará následující část kódu v *edit.js*

```
buttons.edit.click(function () {
    const relayType = $(this).attr('data-config-type'); // Get the relay type from the data attribute
    const nickname = $(this).data('config-nickname'); // Get the nickname from the data attribute

    // Set isBridgeEdit based on the relay type
    isBridgeEdit = relayType === 'bridge';

    const data = {
        nickname: nickname,
        orPort: $(this).data('config-orport'),
        contact: $(this).data('config-contact'),
        controlPort: $(this).data('config-controlport'),
        serverTransport: relayType === 'bridge' ? $(this).data('config-servertransport') : "",
        webtunnelUrl: relayType === 'bridge' ? $(this).data('config-webtunnelurl') : "",
        path: relayType === 'bridge' ? $(this).data('config-path') : "",
        bandwidthRate: $(this).data('config-bandwidthrate'),
    };
```

Obrázek 7.46: Načítání aktuálních hodnot

.data funkce získá hodnoty atributů pro dané jméno služby. Například *$(this).data('config-orport')* získá hodnotu pro *orport* z atributů tlačítka. Uložení atributů tlačítka lze vidět níže.

```
<button class="btn btn-secondary edit-bridge-button"
        th:attr="data-config-type='bridge'"
        th:data-config-contact="${config.getBridgeConfig.contact}"
        th:data-config-controlport="${config.getBridgeConfig.controlPort}"
        th:data-config-nickname="${config.getBridgeConfig.nickname}"
        th:data-config-orport="${config.getBridgeConfig.orPort}"
        th:data-config-servertransport="${config.getBridgeConfig.getServerTransport}"
        th:data-config-webtunnelurl="${config.getBridgeConfig.webtunnelUrl}"
        th:data-config-path="${config.getBridgeConfig.path}"
        th:data-config-bandwidthrate="${config.getBridgeConfig.bandwidthRate}">
    Edit
</button>
```

Obrázek 7.47: Uložení atributů

Po stistknutí tlačítka *save* je odeslán POST požadavek na */update-guard-config* respektive */update-bridge-config*. Endpoint zavolá metodu *updateConfiguration*, která se postará o upravení konfigurace editované služby a vrátí odpověď, zda úprava proběhla úspěšně či ne.

```
public Map<String, String> updateConfiguration(T config) {
    Map<String, String> response = new HashMap<>();
    try {
        String filePath = buildTorrcFilePath(config.getNickname());
        File file = new File(filePath);
        if (file.exists()) {
            boolean deleteResult = file.delete();
            if (!deleteResult) {
                throw new IOException("Failed to delete existing file: " + filePath);
            }
        }
        TorrcFileCreator.createTorrcFile(filePath, config);
        response.put( k: "status",  v: "success");
        response.put( k: "message", getSuccessMessage(config));
    } catch (Exception e) {
        response.put( k: "status",  v: "failure");
        response.put( k: "message",  v: "An unexpected error occurred while updating the configuration.");
    }
    return response;
}
```

Obrázek 7.48: metoda *updateConfiguration*

Zde je také zavolána metoda *createTorrcFile* a od tohoto bodu probíhá stejný postup jako je popsaný v části 7.3.3. Službu lze editovat pouze pokud je zastavena.

- U non-exit uzlu lze upravit **ORPort, Contact email, Control Port** a **Bandwidth limit**.
- U Obfs4 mostu lze upravit **ORPort, ServerTransportListenAddr port, Contact email, Control Port** a **Bandwidth limit**.
- U WebTunnel mostu lze upravit **Contact email** a **Bandwidth limit**.
- Onion služby **nelze editovat**.

7.3.10 Odstraňování služeb

V aplikaci je možné odstranit dříve nakonfigurované služby. To se provede tlačítkem *remove* u příslušné služby. Při mazání je odeslán AJAX POST požadavek na endpoint */relay-operations-api/remove*. Požadavek obrdží kontroler *RelayOperationsRestController*, který jej předá metodě *removeService*. Zde se postupně volají metody na smazání všech příslušných souborů. Při mazání non-exit uzlů a Obfs4 mostů stačí smazat torrc a dataDirectory soubory. K tomu slouží metody *deleteTorrcFile* a *deleteDataDirectory*. Při mazání webTunnelu a onion je navíc

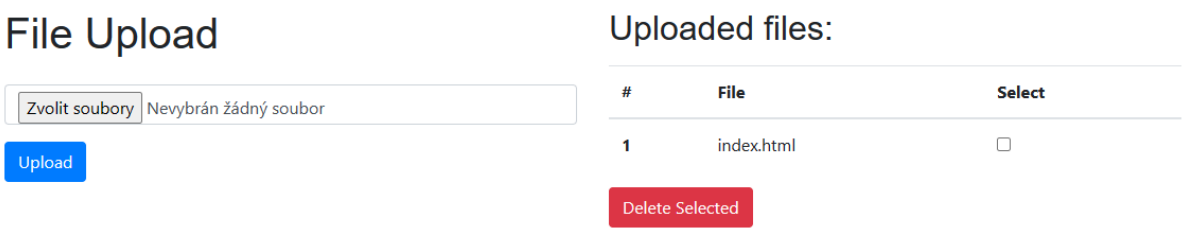
třeba smazat nakonfigurovaný Nginx soubory. Pro to je použita metoda *removeNginxFilesAndConfig*. Tato metoda smaže soubory na webovém serveru a taky symbolický odkaz, který byl vytvořen v */nginx-enabled* složce.

```
public Map<String, Object> removeService(String relayNickname, String relayType) {
    Map<String, Object> response = new HashMap<>();
    try {
        deleteOnionServiceFiles(relayNickname);
        removeNginxFilesAndConfig(relayNickname, relayType);
        deleteDataDirectory(relayNickname, relayType);
        deleteTorrcFile(relayNickname, relayType);

        nginxService.reloadNginx();

        response.put( k: "success",  v: true);
    } catch (IOException | InterruptedException e) {
        response.put( k: "success",  v: false);
    }
    return response;
}
```

Obrázek 7.49: metoda *removeService*

7.3.11 Nahrávání souborů na Onion službu

Na onion služby lze publikovat obsah, který chceme, aby byl dostupný přes síť Tor. K této možnosti se lze dostat kliknutím na tlačítko *edit files* u té služby, přes kterou chceme tento obsah publikovat. To nás dostane na stránku */file/upload/[port]*. Poté se přesuneme na stránku, která se dělí na dvě části. Levá část je pro nahrávání obsahu. Vpravo vidíme obsah nacházející se na této službě a možnost jeho smazání.



Obrázek 7.50: */file/upload/[port]*

Při nahrávání souborů se odešle POST požadavek FileController třídě. Tento kontroler zavolá uploadFiles metodu z FileService třídy, předá nahrané soubory a číslo portu služby, pro který jsou tyto soubory nahrány. Po nahrání vrátí *FileService* seznam jmen všech souborů, které se zde nachází. FileContoller přidá seznam do modelu a vrátí pohled se soubory.

```
@PostMapping("/upload/{port}")
public String uploadFiles(@RequestParam("files") MultipartFile[] files, @PathVariable("port") int port,
                          Model model) {
    try {
        String directory = baseDirectory + port;
        fileService.uploadFiles(files, directory);
        List<String> fileNames = fileService.getUploadedFiles(directory);
        model.addAttribute( attributeName: "uploadedFiles", fileNames);
        model.addAttribute( attributeName: "message",  attributeValue: "Files uploaded successfully!");
    } catch (Exception e) {
        model.addAttribute( attributeName: "message",  attributeValue: "Fail! -> uploaded filename: "
                + Arrays.toString(files));
    }
    return "file_upload_form";
}

 Matys *
@PostMapping("/remove-files/{port}")
public String removeFiles(@RequestParam("selectedFiles") String[] fileNames, @PathVariable("port") int port,
                          Model model) {
    try {
        String directory = baseDirectory + port;
        fileService.deleteFile(fileNames, directory);
        List<String> remainingFileNames = fileService.getUploadedFiles(directory);
        model.addAttribute( attributeName: "uploadedFiles", remainingFileNames);
        model.addAttribute( attributeName: "message",  attributeValue: "Files deleted successfully.");
    } catch (Exception e) {
        model.addAttribute( attributeName: "message",  attributeValue: "Error: " + e.getMessage());
    }
    return "file_upload_form";
}
```

Obrázek 7.51: endpointy pro práci se soubory

Při mazání si uživatel vybere, které soubory si přeje smazat. Odešle se POST požadavek na endpoint */file/remove-files/[port]* se jmény mazaných souborů jako parametr. *FileController* se o tento požadavek postará v *removeFiles* metodě. Soubory se nachází ve složce *onion/www/service-[port]*. Ty soubory, které si přeje uživatel vymazat, jsou přidány k cestě ke složce. Po vymazání všech souborů je opět vrácen seznam se jmény souborů ve složce. Stejný mechanisums nahrávání či mazání souborů funguje i pro WebTunnel mosty, jelikož jsou také hostovány na webovém serveru a lze s tímto mostem na naší stránce publikovat i jiný obsah.

7.3.12 Správa Nginx serveru

Jak bylo již zmíněno, WebTunnel a Onion služby využívají Nginx server. Pro úsporu prostředků na našem Raspberry, se v aplikaci nachází třída *NginxServiceManager*, která hlídá, zda je některá z těchto služeb spuštěna. Dle toho poté řídí, zda má být Nginx server zapnut či vypnut. O to se starají metody *getAllOnionAndWebTunnelServices* a *checkAndManageNginxStatus*. První zmíněná metoda přečte torrc soubory a zjistí, které služby jsou nakonfigurované.

```
public List<String> getAllOnionAndWebTunnelServices() {
    List<String> allServices = new ArrayList<>();
    // Get the list of all onion services
    List<TorConfig> onionConfigs = torConfigService.readTorConfigurations(Constants.TORRC_DIRECTORY_PATH,
            expectedRelayType: "onion");
    for (TorConfig config : onionConfigs) {
        allServices.add(config.getHiddenServicePort());
    }

    // Get the list of all webTunnels
    List<TorConfig> bridgeConfigs = torConfigService.readTorConfigurations(Constants.TORRC_DIRECTORY_PATH,
            expectedRelayType: "bridge");
    for (TorConfig config : bridgeConfigs) {
        allServices.add(config.getBridgeConfig().getNickname());
    }
    return allServices;
}
```

Obrázek 7.52: metoda *getAllOnionAndWebTunnelServices*

Metoda *checkAndManageNginxStatus* poté vezme seznam těchto nakonfigurovaných služeb a zkontroluje jejich status. Pokud alespoň jedna z těchto služeb má status *online*, Nginx server musí být spuštěn.

```java
public void checkAndManageNginxStatus() {
    // Get the list of all webTunnels and Onion services
    List<String> allServices = nginxService.getAllOnionAndWebTunnelServices();

    // Iterate over the list and check the status of each service
    for (String service : allServices) {
        String status = relayStatusService.getRelayStatus(service, relayType: "onion");
        // If at least one service is online, start the Nginx service and return
        if ("online".equals(status)) {
            nginxService.startNginx();
            return;
        }
    }

    // If no service is online, stop the Nginx service
    nginxService.stopNginx();
}
```

Obrázek 7.53: metoda *checkAndManageNginxStatus*

7.3.13 UPnP

Aplikace zahrnuje možnost na stránce s nakonfigurovanými službami zapnout automatické otevírání/zavírání portů, pokud je na síti dostupný UPnP router. K tomuto účelu je využita *WaifUPnP* knihovna [25]. Nejprve se odešle GET požadavek na */relay-operations-api/upnp-availability* endpoint, aby se zjistilo, zda je dostupný UPnP router. K tomu v knihovně slouží metoda *isUPnPAvailable*. Porty, které je potřeba otevřít, jsou získány metodou *getPorts*, která přečte torrc soubory a extrahuje porty, které je třeba otevřít. Pro non-exit uzly to je **ORPort**, pro Obfs4 most **ServerTransportListenAddr port a ORPort** a pro WebTunnel **WebTunnel port**.

```
while ((line = reader.readLine()) != null) {
    if (line.startsWith("ORPort")) {
        if (!line.equals("ORPort 127.0.0.1:auto IPv4Only")) {
            String portStr = line.split( regex: " ")[1];
            ports.get("ORPort").add(Integer.parseInt(portStr));
        }
    }
    if (line.startsWith("ServerTransportListenAddr obfs4")) {
        String[] parts = line.split( regex: " ");
        if (parts.length > 2) {
            String[] addrParts = parts[2].split( regex: ":");
            if (addrParts.length > 1) {
                ports.get("Obfs4Port").add(Integer.parseInt(addrParts[1]));
            }
        }
    }
    if (line.startsWith("# webtunnel")) {
        ports.get("WebTunnelPort").add(Integer.parseInt(line.split( regex: " ")[2]));
    }
```

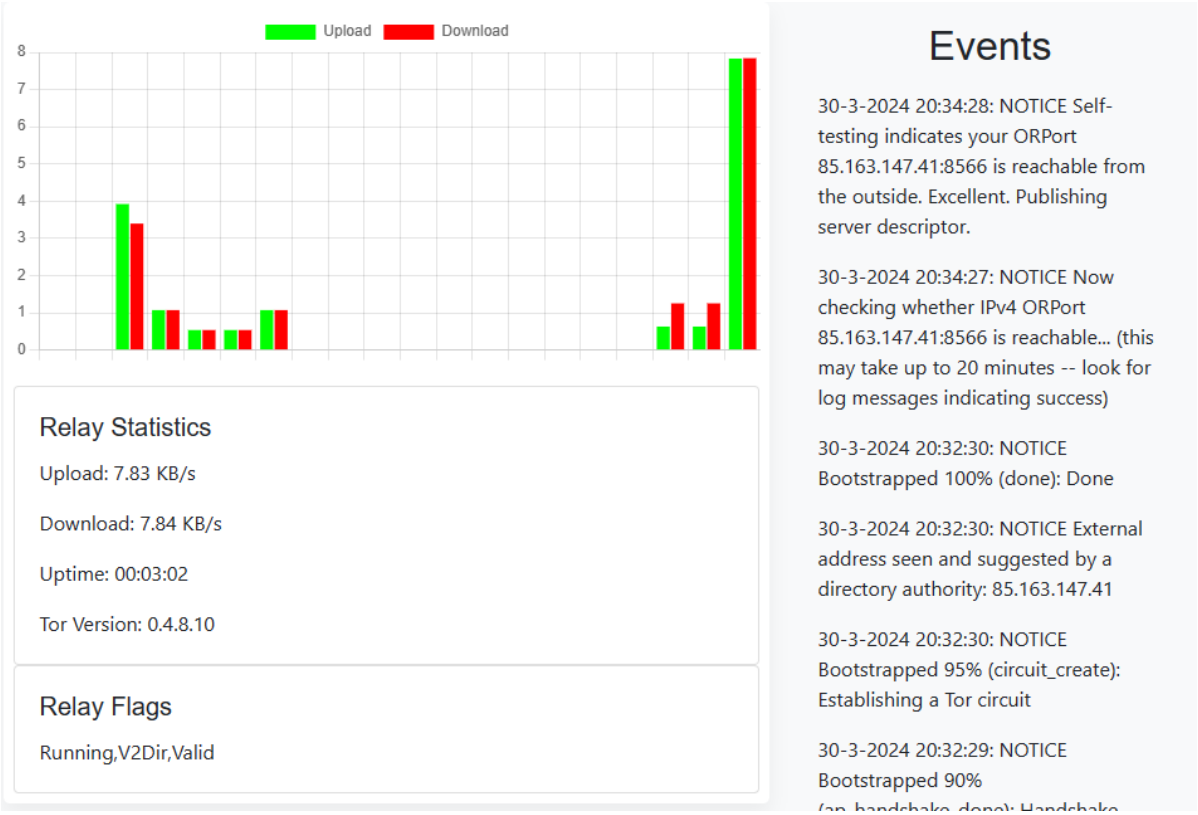
Obrázek 7.54: Získání potřebných portů

Porty se otevírají a zavírají dle toho, zda je daná služba online či offline. Kontrola probíhá prvně při zapnutí automatického otevírání portů, kdy se zkontroluje, které služby jsou online a těm se otevřou porty. Poté probíhá změna v moment, kdy se služba vypíná respektive zapíná.

```
for (TorConfig config : allConfigs) {
    String relayType = null;
    String relayNickname = null;
    if (config.getGuardConfig() != null) {
        relayType = "guard";
        relayNickname = config.getGuardConfig().getNickname();
    } else if (config.getBridgeConfig() != null) {
        relayType = "bridge";
        relayNickname = config.getBridgeConfig().getNickname();
    }

    if (relayType != null && relayNickname != null) {
        if (enable) {
            String status = relayStatusService.getRelayStatus(relayNickname, relayType);
            if ("online".equals(status)) {
                openPorts(relayNickname, relayType);
            }
        } else {
            closePorts(relayNickname, relayType);
        }
    }
}
```

Obrázek 7.55: Otevírání portů při zapnutí automatického otevírání portů

7.3.14 Sledování provozu



Obrázek 7.56: Traffic data stránka

Na stránce */traffic-data* je možno sledovat provoz a události na jednotlivých službách. K tomuto účelu byla použita python knihovna **stem**[26]. Ke sledování provozu Tor služeb existuje program **Nyx**, který právě tuto knihovnu také využívá[38]. Nyx běží v příkazovém řádku a byl hlavní inspirací pro přetvoření některých jeho funkcí do naší aplikace.

Obrázek 7.57: Program Nyx [38]

Funkcionalita pro monitorování provozu se nachází v *data.py*. Skript postupně v cyklu prochází torrc soubory a skenuje je pro *Control porty*. Pro každý control port je vytvořeno nové vlákno, které se stará o monitorování daného *control portu*. Skenování se opakuje každých 10 sekund.

```
# Iterate through the files in the directory and collect control ports
for filename in os.listdir(torrc_dir):
    control_port = read_control_port(os.path.join(torrc_dir, filename))
    if control_port and control_port not in control_ports:
        control_ports.append(control_port)
        thread = threading.Thread(target=monitor_traffic_and_flags, args=(control_port,))
        thread.daemon = True  # Set the thread as a daemon
        threads.append(thread)
        thread.start()
```

Obrázek 7.58: Získávání control portů a jejich monitorování

Samotné čtení *control portů* probíhá ve funkci *read_control_port*. Otevře soubor a načte řádek *ControlPort*

```python
def read_control_port(file_path):
    """
    Read the control port from the Tor configuration file.

    :param file_path: Path to the Tor configuration file.
    :return: Control port number if found, None otherwise.
    """
    try:
        with open(file_path, 'r') as file:
            for line in file:
                if line.startswith("ControlPort"):
                    parts = line.strip().split()
                    if len(parts) == 2:
                        return int(parts[1])
    except FileNotFoundError:
        print(f"File not found: {file_path}")
    except Exception as e:
        print(f"Error reading control port from {file_path}: {str(e)}")
    return None
```

Obrázek 7.59: funkce *read_control_port*

Funkce *monitor_traffic_and_flags* monitoruje provoz a vlajky (flagy) na uzlech. Existuje několik vlajek. Tyto vlajky jsou přidělovány uzlům a informují o jejich chování či vlastnostech. Mezi hlavní vlajky patří:

- **BadExit**: Pokud je uzel považován za nepoužitelný jako výstupní uzel. Například jej může cenzurovat poskytovatel internetu.
- **Exit**: Označení výstupního uzlu.
- **Fast**: Pokud je uzel vhodný pro sestavení okruhu s velikou rychlostí.
- **Guard**: Pokud je uzel vhodný použít jako vstupní uzel do Toru.
- **HSDir**: Pokud je uzel považován za skrytý adresář služeb. To znamená, že se zde mohou ukládat informace o onion službách.
- **V2Dir**: Uzel podporující v2 directory protocol nebo vyšší [39].
- **Stable**: Pokud uzel běží dostatečnou dobu bez přerušení.
- **Running**: Značí, že je uzel použitelný.
- **Valid**: Pokud uzel běží na podporované verzi Toru a nebyl označen jako podezřelý.

Funkce se pomocí *Controller.from_port(port=control_port)* metody ze *Stem* knihovny připojí ke *control portu*, ověří spojení a povolí údálosti od úrovně *NOTICE* a vyšší.

```
controller.authenticate()
controller.set_conf('__LeaveStreamsUnattached', '1')
controller.set_conf('ControlPort', str(control_port))
controller.set_conf('HashedControlPassword', '')
controller.set_conf('CookieAuthentication', '1')
controller.set_conf('Log', ['NOTICE stdout'])
```

Obrázek 7.60: Nastavení konfigurace při připojení ke *control portu*

Data jsou posílána na endpoint *relay-data/relays/[control_port]* pomocí funkce *_send_bandwidth_data*, která načte celkový počet bajtů přečtených a zapsaných z controlleru, což představuje data pro stahování a odesílání. Také přečte vlajky, celkový uptime a kterou Tor verzi uzel používá. Všechna tato data jsou pak zabalena a odeslána na endpoint API spojený s control portem. Funkce používá metodu requests.post k odeslání dat jako objektu JSON.


```
{
    "download": 0,
    "upload": 6212,
    "bandwidth": 0,
    "uptime": 9954,
    "flags": [
        "Running",
        "V2Dir",
        "Valid"
    ],
    "event": null,
    "version": "0.4.8.10"
},
{
    "download": 0,
    "upload": 0,
    "bandwidth": 0,
    "uptime": 9960,
    "flags": [
        "Running",
        "V2Dir",
        "Valid"
    ],
    "event": null,
    "version": "0.4.8.10"
},
{
    "download": 0,
    "upload": 0,
    "bandwidth": 0,
    "uptime": 9966,
    "flags": [
        "Running",
        "V2Dir",
        "Valid"
    ],
    "event": null,
    "version": "0.4.8.10"
},
```


Obrázek 7.61: endpoint *relay-data/relays/[control_port]*

```
# Get the initial total bytes read and written
initial_download = int(controller.get_info("traffic/read"))
initial_upload = int(controller.get_info("traffic/written"))

# Wait for 1 second
time.sleep(1)

# Get the total bytes read and written after 1 second
final_download = int(controller.get_info("traffic/read"))
final_upload = int(controller.get_info("traffic/written"))

# Calculate the per-second rates
download_rate = final_download - initial_download
upload_rate = final_upload - initial_upload

# Get the relay flags
flags = relay_flags(controller)

# Get the uptime
uptime = int(controller.get_info("uptime"))

# Get the Tor version
tor_version = controller.get_info("version")
```

Obrázek 7.62: Získání dat o uzlu

```
# Create a dictionary with the bandwidth data and an identifier
data = {
    "download": download_rate,
    "upload": upload_rate,
    "flags": flags,  # Add the flags to the data
    "uptime": uptime,  # Add the uptime to the data
    "version": tor_version,  # Add the Tor version to the data
}

# Construct the complete API endpoint URL with the relayId
api_endpoint = f"{BASE_API_ENDPOINT}/{control_port}"

# Send data to the API endpoint for the corresponding relay
response = requests.post(api_endpoint, json=data, verify=False)
```

Obrázek 7.63: Odeslání dat

K odesílání událostí slouží *_handle_event* funkce, která události posílá na *relay-data/relays/[control_por*
endpoint.

```python
# Create a dictionary with the event data and an identifier
data = {
    "event": str(event),
}

# Construct the complete API endpoint URL with the relayId
api_endpoint = f"{BASE_API_ENDPOINT}/{control_port}/event"

# Send data to the API endpoint for the corresponding relay
requests.post(api_endpoint, json=data, verify=False)
```

Obrázek 7.64: Odeslání dat o událostech

Data odeslaná na endpoint zpracuje **RelayDataRestController** pomocí metod *createRelayData* a *createRelayData*. Údaje jsou uloženy do *relayDataMap* a *relayEventMap* a jsou použity k aktualizaci údajů zobrazovaných na webové stránce. Skript *data.js* učiní GET požadavek, aby tato data načetl.

```java
@PostMapping("/relays/{relayId}")
public ResponseEntity<String> createRelayData(@PathVariable int relayId, @RequestBody RelayData relayData) {
    relayDataService.handleRelayData(relayId, relayData, relayDataMap);
    return ResponseEntity.ok( body: "Data received successfully for Relay ID: " + relayId);
}


Matys +1
@PostMapping("/relays/{relayId}/event")
public ResponseEntity<String> createRelayEvent(@PathVariable int relayId, @RequestBody Map<String,
        String> eventData) {
    relayDataService.handleRelayEvent(relayId, eventData, relayDataMap, relayEventMap);
    return ResponseEntity.ok( body: "Event received successfully for Relay ID: " + relayId);
}
```

Obrázek 7.65: POST endpointy v *RelayDataRestController* třídě

```
$.get(apiUrl, function (data) : void {
    if (data && data.length > 0) {
        // Filter out null values from the data
        data = data.filter(function (relayData) : boolean {
            return relayData !== null;
        });

        let uploadData = data.map(function (relayData) : XMLHttpRequestUpload {
            return relayData.upload;
        });
        let downloadData = data.map(function (relayData) : string {
            return relayData.download;
        });

        const flagsData = data.map(function (relayData) : string {
            return relayData.flags;
        });
```

Obrázek 7.66: Načítání dat v *data.js*

Na endpoint */relay-data/info* jsou metodou *getRelayInfo* poslány některé údaje (přezdívka uzlu, control port, typ). Z tohoto endpointu jsou načtena data GET požadavkem v *data.js*, který jej použije pro sestavení menu, ze kterého si uživatel vybírá uzel na jehož provoz se chce podívat.

```
public List<RelayInfo> getRelayInfo() {
    List<RelayInfo> relayInfoList = new ArrayList<>();

    // Fetch the list of all bridges
    List<BridgeConfig> bridges = relayInformationService.getAllBridges();
    for (BridgeConfig bridge : bridges) {
        RelayInfo relayInfo = new RelayInfo(Integer.parseInt(bridge.getControlPort()), bridge.getNickname(),
                type: "bridge");
        relayInfoList.add(relayInfo);
    }

    // Fetch the list of all guards
    List<GuardConfig> guards = relayInformationService.getAllGuards();
    for (GuardConfig guard : guards) {
        RelayInfo relayInfo = new RelayInfo(Integer.parseInt(guard.getControlPort()), guard.getNickname(),
                type: "guard");
        relayInfoList.add(relayInfo);
    }

    return relayInfoList;
}
```

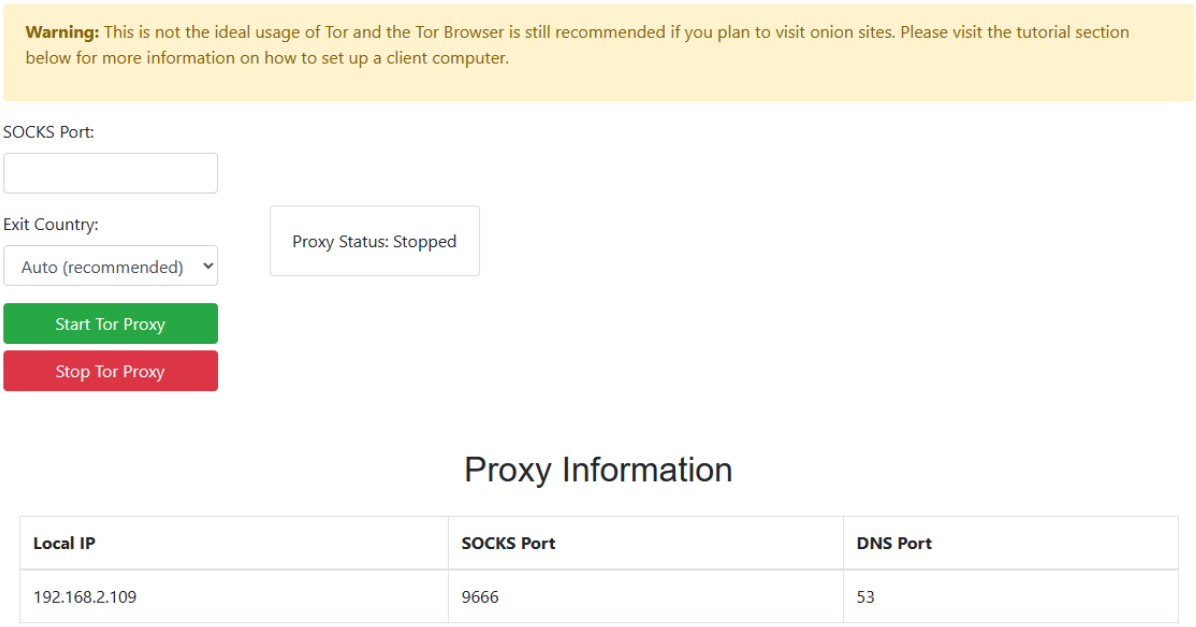
Obrázek 7.67: Metoda *getRelayInfo*

Obrázek 7.68: endpoint */relay-data/info*

Obrázek 7.69: Menu pro výběr sledovaného uzlu

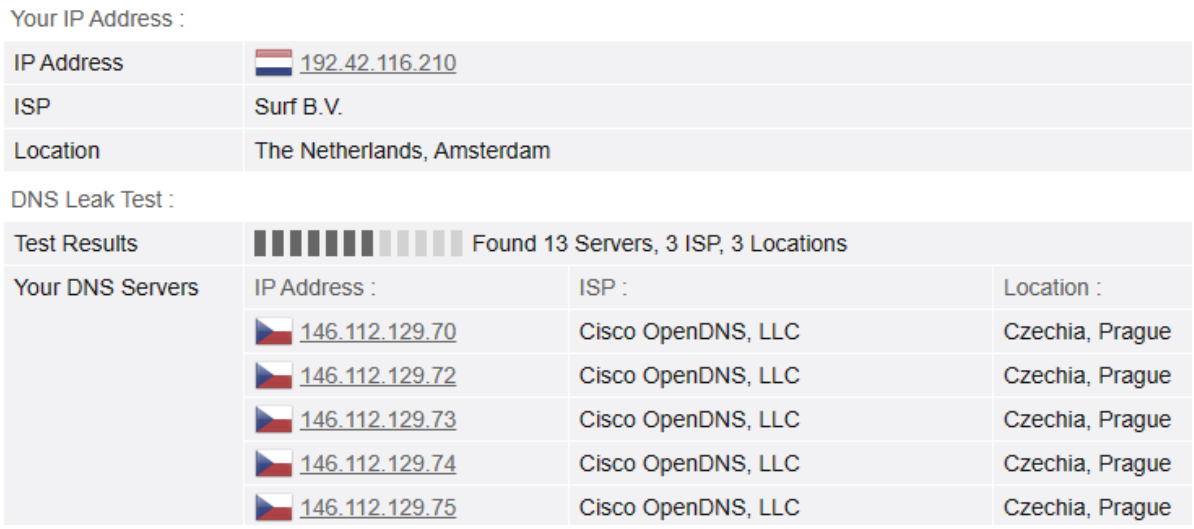
7.3.15 Proxy server

Poslední částí aplikace je stránka */proxy*, kde si uživatel může nastavit proxy server. Tor umožňuje směrování komunikace přes **SOCKS5** proxy. Při použití proxy **SOCKS5** jsou datové pakety z nakonfigurovaného zdroje směrovány přes vzdálený server. Tento server změní IP adresu přidruženou k těmto datovým paketům předtím, než dosáhnou svého konečného cíle, což nabízí větší anonymitu online. Díky tomuto je možné posílat komunikaci přes Tor i v jiných aplikacích, než je Tor prohlížeč. Tor v tento moment podporuje pouze komunikaci přes TCP. Na UDP komunikaci se pracuje.



Obrázek 7.70: Stránka /proxy

Na této stránce si uživatel zvolí port, na kterém chce **SOCSK5** proxy provozovat. Lze si zde také zvolit preferovaný stát, ze kterého uživatel chce, aby jeho provoz vycházel. Tohle je jediná část aplikace, kde je pro plnou funkcionalitu potřeba nastavit pár věcí na klientském PC. Z tohoto důvodu se zde zobrazuje lokální IP adresa Raspberry Pi a DNS port pro snadnější konfiguraci. Na dolní části stránky se nachází návody na správnou konfiguraci klientského PC. Je potřeba nastavit **SOCKS5** proxy na lokální IP adresu Raspberry Pi a na nastavený port. Na stejnou IP adresu je třeba nastavit i DNS, aby se zabránilo DNS úniku.



Obrázek 7.71: Používání Tor proxy před ošetření DNS leaku

Your IP Address :

IP Address	192.42.116.210		
ISP	Surf B.V.		
Location	The Netherlands, Amsterdam		

DNS Leak Test :

Test Results	Found 4 Servers, 1 ISP, 1 Location		
Your DNS Servers	IP Address :	ISP :	Location :
	217.197.80.4	Individual Network Berlin e.V.	Germany, Berlin
	217.197.80.5	Individual Network Berlin e.V.	Germany, Berlin
	2001:67c:1400:1010::4	Individual Network Berlin e.V.	Germany, Berlin
	2001:67c:1400:1010::5	Individual Network Berlin e.V.	Germany, Berlin

Obrázek 7.72: Používání Tor proxy po ošetření DNS leaku

Další věc, která může prozradit IP adresu, je WebRTC. WebRTC je technologie, která umožňuje přímou peer-to-peer komunikaci mezi prohlížeči, například videohovory, hlasové chaty a sdílení souborů. WebRTC používá techniku nazvanou "ICE"(Interactive Connectivity Establishment) k nalezení nejlepší cesty pro přenos dat mezi dvěma zařízeními. Součástí procesu ICE je výměna informací o síti, včetně IP adresy. Proto se na stránce též nachází doporučení, jak lze WebRTC vypnout. Pro chromium prohlížeče je většinou potřeba stáhnout rozšíření, jelikož se v nich zpravidla WebRTC přímo vypnout nedá. V aplikaci je doporučeno rozšíření **WebRTC Control**[40]. Ve Firefox prohlížeči se dá WebRTC vypnout v nastavení.

Samotné nastavování proxy serveru probíhá podobným způsobem, jako konfigurace ostatních služeb. Je třeba vytvořit příslušný torrc soubor se zvoleným nastavením. Pro konfiguraci proxy slouží metoda *initializeAndRunProxy*. Ta postupně volá metody, které jsou při konfiguraci použity. Opět se provede kontrola dostupnosti portu pomocí *portsAreAvailable*. Pokud je port dostupný, zavolá se metoda *setupProxyConfiguration*, která vytvoří *torrc* soubor pro proxy server.

```
bw.write( str: "SocksPort " + localIpAddress + ":" + socksPort);
bw.newLine();
String localNetwork = localIpAddress.substring(0, localIpAddress.lastIndexOf( ch: '.')) + ".0/24";
bw.write( str: "SocksPolicy accept " + localNetwork);
bw.newLine();
bw.write( str: "RunAsDaemon 1");
bw.newLine();
bw.write( str: "DNSPort " + localIpAddress + ":53");
bw.newLine();
if (!"auto".equals(exitCountry)) {
    bw.write( str: "ExitNodes {" + exitCountry + "}");
    bw.newLine();
    bw.write( str: "StrictNodes 1");
}
```

Obrázek 7.73: Tvorba Tor proxy *torrc* souboru

```
SocksPort 192.168.2.126:2500
SocksPolicy accept 192.168.2.0/24
RunAsDaemon 1
DNSPort 192.168.2.126:53
ExitNodes {en}
StrictNodes 1
```

Obrázek 7.74: *Torrc* proxy serveru

Torrc obsahuje následující řádky:

- **SocksPort** určuje, na kterém portu bude Tor naslouchat pro SOCKS5 požadavky. IP adresa
- **SocksPolicy** definuje, které IP adresy budou moci používat proxy server Tor. V tomto případě budou moci proxy server Tor používat všechny počítače v podsíti 192.168.2.0/24.
- **RunAsDaemon** jsme si již vysvětlovali. Spustí službu na pozadí.
- **DNSPort** určuje port, na kterém bude Tor naslouchat pro DNS dotazy.

Pokud si uživatel při konfiguraci vybere stát, který chce použít pro výstup z Tor sítě, jsou do *torrc* souboru vloženy poslední dva řádky.

- **ExitNodes** určuje, z jakých zemí bude provoz z Toru vycházet.

- **StrictNodes** nutí Tor použít námi specifikované uzly.

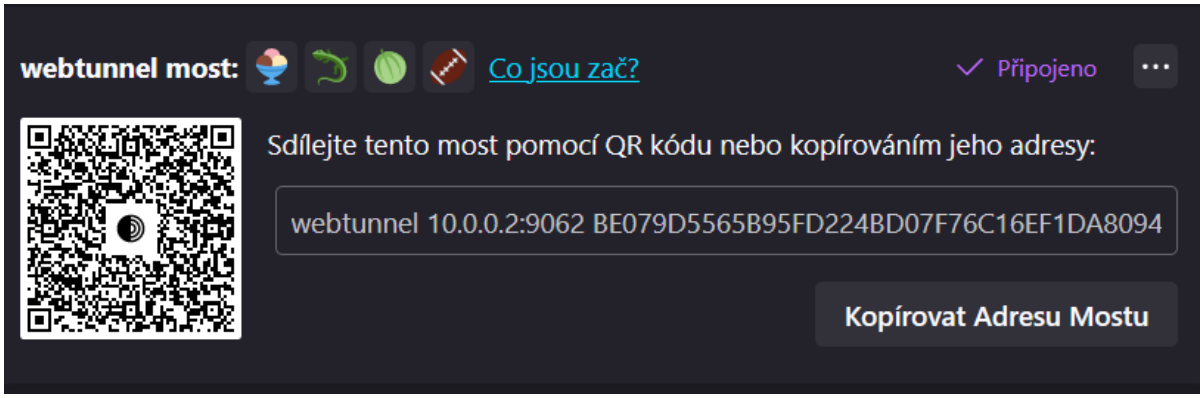
Po konfiguraci je zavolána metoda *startProxy*, která nastartuje proxy.

```
public long startProxy(String filePath) throws IOException, InterruptedException {
    long pid = relayStatusService.getTorRelayPID(filePath);
    if (pid != -1) {
        return pid;
    }

    ProcessBuilder processBuilder = new ProcessBuilder( ...command: "/bin/bash", "-c", "sudo tor -f " + filePath);
    processBuilder.redirectErrorStream(true);
    Process process = processBuilder.start();
    try {
        int exitCode = process.waitFor();
        if (exitCode == 0) {
            pid = process.pid();
            return pid;
        }
    } finally {
        process.destroy();
    }
    return -1;
}
```

Obrázek 7.75: Metoda *startProxy*

7.4 Testování

Platforma byla testována pravidelně testována během vývoje aplikace. Při přidání možnosti konfigurace nové služby byla tato služba otestována ve všech částech od konfigurace, přes upravování až po spuštění a následné monitorování. Pokaždé byl proveden test stability dané služby, kdy jsem Raspberry nechal běžet delší dobu se spuštěnou službou a sledoval, zda se během této doby nic nepokazí. Tato doba většinou trvala přibližně 24 hodin. Při testování non-exit uzlů byla zkontrolována jejich dostupnost pro síť Tor na stránce *https://metrics.torproject.org/*. Pro otestování mostů byl zkopírován odkaz těchto mostů a následně jsem se z počítače s nainstalovaným Tor prohlížečem z jiné sítě pokusil k tomuto mostu připojit. Připojení bylo úspěšné a tím byla otestována správnost konfigurace mostu, vytváření odkazů a samotná dostupnost.



Obrázek 7.76: Testování dostupnosti mostu

Pro otestování onion služeb stačilo zkopírovat onion odkaz a vložit jej do prohlížeče Tor, kde se nám daná onion služba načetla. Testování funkčnosti proxy proběhlo změnou DNS a Proxy nastavení v systému Windows a následnou kontrolou zobrazované IP adresy a DNS záznamů na stránce *https://browserleaks.com*. Bylo též otestováno mazání služeb a kontrola souborů, zda za sebou mazání nic nezanechává. V době psaní této práce jsou na Raspberry Pi provozovány dva non-exit uzly a jedna onion služba.

8. Závěr

Tato bakalářská práce demonstruje vytvoření webové aplikace pro konfiguraci a správu Tor služeb pomocí počítače Raspberry Pi s operačním systémem Raspbian. Vyvinutá aplikace umožňuje uživatelům snadno konfigurovat, monitorovat a upravovat služby Tor a zároveň poskytuje funkce proxy serveru SOCKS5 pro zařízení, která nemohou přímo nainstalovat prohlížeč Tor.

Aplikace by do budoucna mohla být rozšířena o podporu IPv6, jelikož nyní aplikce podporuje pouze IPv4, která je ale potřeba i při provozu přes IPv6. Další možností by byla podpora exit uzlů. Toto by bylo potřeba ovšem velice dobře promyslet a uvědomit si možná rizika, která by to pro uživatele představovalo. Jednou z možností, kterou bych do budoucna rád přidal, by byla bezpečná extrakce a následné importování klíčů uzlů, které slouží k obnově reputace uzlu při jeho opětovné konfiguraci například při selhání disku. Při tomto procesu se musí postupovat opatrně, jelikož při úniku klíčů může dojít k dešifrování provozu na uzlu či ukradení jeho totožnosti.

Seznam použité literatury

[1] The Layers of the Web – Surface Web, Deep Web and Dark Web [online]. [cit. 2024-02-26]. Dostupné z: https://ifflab.org/the-layers-of-the-web-surface-web-deep-web-and-dark-web/

[2] Tor Browser - Download [online]. [cit. 2024-02-26]. Dostupné z: https://www.torproject.org/download/

[3] The Layers of the Web – Surface Web, Deep Web and Dark Web. [cit. 2024-02-26]. Dostupné z: https://ifflab.org/the-layers-of-the-web-surface-web-deep-web-and-dark-web/

[4] Am I Unique? [cit. 2024-02-26]. Dostupné z: https://amiunique.org/

[5] Browser Fingerprinting: An Introduction and the Challenges Ahead [cit. 2024-02-26]. Dostupné z: https://blog.torproject.org/browser-fingerprinting-introduction-and-challenges-ahead/

[6] ABOUT TOR BROWSER [cit. 2024-02-26]. Dostupné z: https://tb-manual.torproject.org/about/

[7] Help Censored Users, Run a Tor Bridge [cit. 2024-02-26]. Dostupné z: https://blog.torproject.org/run-a-bridge-campaign/

[8] Wrapping up Run a Tor bridge campaign [cit. 2024-02-26]. Dostupné z: https://blog.torproject.org/wrapping-up-bridges-campaign/

[9] Tor Metrics [cit. 2024-02-26]. Dostupné z: https://metrics.torproject.org/

[10] Bryan Ford. Peer-to-Peer Communication Across Network Address Translators [cit. 2024-02-27]. Dostupné z: https://bford.info/pub/net/p2pnat/

[11] How do Onion Services work? [cit. 2024-02-27]. Dostupné z: https://community.torproject.org/onion-services/overview/

[12] Tor Specifications - Encoding onion addresses [ONIONADDRESS] [cit. 2024-02-27]. Dostupné z: https://spec.torproject.org/rend-spec/encoding-onion-addresses.html

[13] V2 Onion Services Deprecation [cit. 2024-02-27]. Dostupné z: https://support.torproject.org/onionservices/v2-deprecation/

[14] Relay requirements [cit. 2024-02-28]. Dostupné z: https://community.torproject.org/relay/relays-requirements/

[15] Cyril Šebek. Lekce 1 - Úvod do Raspberry Pi [cit. 2024-02-28]. Dostupné z: https://www.itnetwork.cz/hardware-pc/raspberry-pi/uvod-do-raspberry-pi

[16] RPishop.cz [cit. 2024-02-29]. Dostupné z: https://rpishop.cz/

[17] Raspberry Pi USB-C 5,1V=3A napájecí zdroj, EU, bílý. RPishop.cz [cit. 2024-02-29]. Dostupné z: https://rpishop.cz/zdroje-s-usb-c-kabelem/1594-ofi046.html

[18] Xiaomi Original USB nabíječka (33W) bílá (OEM). smarty.cz [cit. 2024-02-29]. Dostupné z: https://www.smarty.cz/Xiaomi-Original-USB-nabijecka-33W-bila-OEM–p144801

[19] Argon NEO Raspberry Pi 4 Case. rpishop.cz [cit. 2024-02-29]. Dostupné z: https://rpishop.cz/raspberry-pi-4/2069-argon-neo-raspberry-pi-4-case.html

[20] Operating system images. Raspberry Pi [cit. 2024-02-29]. Dostupné z: https://www.raspberrypi.com/software/operating-systems/

[21] Dark net raids were 'overblown' by police, says Tor Project. BBC [cit. 2024-03-01]. Dostupné z: https://www.bbc.com/news/technology-29987379

[22] MAREK JANOUŠ. Provozoval uzel sítě Tor, a obvinili jej z šíření dětské pornografie [cit. 2024-03-01]. Dostupné z: https://www.lupa.cz/clanky/tor-zasah/

[23] Tor Project. Good Bad ISPs [cit. 2024-03-01]. Dostupné z: https://community.torproject.org/relay/community-resources/good-bad-isps/

[24] Thymeleaf [cit. 2024-03-03]. Dostupné z: https://www.thymeleaf.org/

[25] Federico Dossena. WaifUPnP [cit. 2024-03-03]. Dostupné z: https://github.com/adolfintel/WaifUPnP

[26] Stem [cit. 2024-03-04]. Dostupné z: https://stem.torproject.org/

[27] The Software Development Kit Manager [cit. 2024-03-04]. Dostupné z: https://sdkman.io/

[28] Acme.sh [cit. 2024-03-06]. Dostupné z: https://github.com/acmesh-official/acme.sh

[29] WEDOS [cit. 2024-03-09]. Dostupné z: https://www.wedos.com/cs/

[30] WebTunnel Bridge [cit. 2024-03-09]. Dostupné z: https://community.torproject.org/relay/setup/webtunnel/

[31] Tor Relay Configurator [cit. 2024-03-09]. Dostupné z: https://tor-relay.co/

[32] Felix Stein. tor-relay-configurator [cit. 2024-03-09]. Dostupné z: https://github.com/flxn/tor-relay-configurator

[33] OnionShare [cit. 2024-03-09]. Dostupné z: https://docs.onionshare.org/2.6.2/en/

[34] abysshint. tor-control-panel [cit. 2024-03-09]. Dostupné z: https://github.com/abysshint/tor-control-panel?tab=readme-ov-file

[35] Let's Encrypt [cit. 2024-03-09]. Dostupné z: https://letsencrypt.org/

[36] Relay requirements [cit. 2024-03-16]. Dostupné z: https://community.torproject.org/relay/relays-requirements/

[37] Shelikhoo, Ggus. Hiding in plain sight: Introducing WebTunnel [cit. 2024-03-16]. Dostupné z: https://blog.torproject.org/introducing-webtunnel-evading-censorship-by-hiding-in-plain-sight/

[38] Tor Nyx [cit. 2024-03-24]. Dostupné z: https://nyx.torproject.org/

[39] Tor directory protocol, version 3, Tor Specifications [cit. 2024-03-24]. Dostupné z: https://spec.torproject.org/dir-spec/

[40] Grephz, WebRTC Control [cit. 2024-03-24]. Dostupné z: https://chromewebstore.google.com/detail/webrtc-control/fjkmabmdepjfammlpliljpnbhleegehm

Seznam obrázků

2.1 Vrstvy internetu [3] 5
2.2 Výsledek testu na stránce *AmIUnique.org* [4] 6
2.3 Tor prohlížeč při spuštění 7
2.4 Počet mostů po "Run a Tor Bridge"kampani [8] 9
2.5 Podíl uživatelů připojujících se k mostům [9] 9
2.6 Graf vývoje užívání mostů uživateli z Ruska [9] 10
2.7 Torrc soubor 13
2.8 Přehled některých českých poskytovatelů internetu a jejich postoj k provozu Tor uzlů [23] 14

3.1 *tor-relay.co*[**?**] 17
3.2 Upozornění na stránce *tor-relay.co* 18
3.3 OnionShare[33] 18
3.4 Tor control panel [34] 19

4.1 Architektura webové aplikace 21

5.1 Raspberry Pi 4 Model B [15] 23

7.1 Java verze dostupné přes SDKMan 27
7.2 Hešování přihlašovacích údajů 32
7.3 Balíčky aplikace 32
7.4 Struktura webové části aplikace 33
7.5 Domovská obrazovka 34
7.6 Setup obrazovka 35
7.7 */guard/configure* endpoint 36
7.8 *configureGuard* metoda 36
7.9 *createTorrcFile* metoda 37
7.10 *writeTorrcFileConfig* metoda 37
7.11 Torrc soubor non-exit uzlu 38
7.12 Nastavení mostu 39

7.13 */bridge/configure* endpoint . . . 39
7.14 Obfs4 Torrc soubor . . . 40
7.15 Metoda *setupSnowflakeProxy* . . . 41
7.16 Nastavení Snowflake proxy . . . 41
7.17 Kód nastavující webový server pro provoz WebTunnelu . . . 42
7.18 metoda generateCertificate . . . 43
7.19 metoda createCertInstallationProcessBuilder . . . 44
7.20 Konfigurace default Nginx konfiguračního souboru . . . 45
7.21 WebTunnel torrc soubor . . . 45
7.22 Nastavení onion služby . . . 46
7.23 */onion-service/configure endpoint* . . . 46
7.24 *buildNginxServerBlock* metoda . . . 47
7.25 *deployOnionServiceNginxConfig* metoda . . . 47
7.26 torrc soubor služby onion . . . 48
7.27 getGuardCount metoda . . . 48
7.28 getGuardCount metoda . . . 49
7.29 Varování, které uživatele informuje, že již nakonfiguroval non-exit uzel . . . 50
7.30 Nakonfigurovaný uzel . . . 50
7.31 Zpracování Torrc souborů v metodě *parseTorConfig* . . . 51
7.32 Přidání TorConfig objektů do modelu . . . 52
7.33 Generování HTML pomocí Thymeleaf pro guard uzly . . . 52
7.34 Nakonfigurované Obfs4 a WebTunnel mosty . . . 53
7.35 metoda *getWebtunnelLink* . . . 54
7.36 metoda *readHostnameFile* . . . 54
7.37 Nakonfigurovaná Onion služba . . . 55
7.38 Ajax požadavek na spuštění služby . . . 55
7.39 Metoda *startRelay* . . . 56
7.40 Příkaz pro spuštění služby . . . 56
7.41 Metoda *getTorRelayPID* . . . 57
7.42 Metoda *getGuardRelayFingerprints* . . . 58
7.43 Metoda *getRelayStatus* . . . 58

7.44 *status.js* 59
7.45 Editační okno pro Obfs4 most 59
7.46 Načítání aktuálních hodnot 60
7.47 Uložení atributů 60
7.48 metoda *updateConfiguration* 61
7.49 metoda *removeService* 62
7.50 */file/upload/[port]* 62
7.51 endpointy pro práci se soubory 63
7.52 metoda *getAllOnionAndWebTunnelServices* 64
7.53 metoda *checkAndManageNginxStatus* 65
7.54 Získání potřebných portů 66
7.55 Otevírání portů při zapnutí automatického otevírání portů 66
7.56 Traffic data stránka 67
7.57 Program Nyx [38] 68
7.58 Získávání control portů a jejich monitorování 68
7.59 funkce *read_control_port* 69
7.60 Nastavení konfigurace při připojení ke *control portu* 70
7.61 endpoint *relay-data/relays/[control_port]* 70
7.62 Získání dat o uzlu 71
7.63 Odeslání dat 71
7.64 Odeslání dat o událostech 72
7.65 POST endpointy v *RelayDataRestController* třídě 72
7.66 Načítání dat v *data.js* 73
7.67 Metoda *getRelayInfo* 73
7.68 endpoint */relay-data/info* 74
7.69 Menu pro výběr sledovaného uzlu 74
7.70 Stránka /proxy 75
7.71 Používání Tor proxy před ošetření DNS leaku 75
7.72 Používání Tor proxy po ošetření DNS leaku 76
7.73 Tvorba Tor proxy *torrc* souboru 77
7.74 *Torrc* proxy serveru 77

7.75 Metoda *startProxy* 78
7.76 Testování dostupnosti mostu 79

8.1 Raspberry Pi počítač v Argon NEO krabičce 88

Přílohy

Přílohy v přiloženém komprimovaném souboru:

- Kompletní kód aplikace
- Složka s připravenou aplikací pro běh a instrukcemi pro její spuštění.

Kód spolu se sestavenou aplikací je též dostupný na githubu.

https://github.com/Matys134/torConfigTool

Aplikace pro hašování přihlašovacích údajů.

https://github.com/Matys134/userHasherApplication

Raspberry Pi počítač, pro který byla aplikace vyvíjena.

Obrázek 8.1: Raspberry Pi počítač v Argon NEO krabičce