概要

インターネット依存に対処し, 健康的な生活を送るために, インターネット利用を時間でコントロールする HTTP(S) プロキシを実装した.

本プロキシの開発に当たり, 医療の現場で行われているインターネット依存の治療法の一つである, 「外的ストップテクニック」と呼ばれる, 一定時間が経過したらオンライン活動をストップする手法を参考にした. しかし現代社会において, オンライン活動をすべて遮断してしまうことは, 必要な連絡を受け取れないといった不都合を生み出してしまう. そこで, 依存している SNS サービスなどのブロックしたいアドレスを事前に登録しておき, アクセスしてから一定時間が経過したら通信を遮断, また一定時間後に通信を再開するプロキシを実装した.

実装後, LAN 内にプロキシサーバを建てて運用する中で, 確かに SNS 利用が抑えられ, 目の前のタスクに集中できた感触があった. 今後は, プロキシサービスを運用し, 被験者の数を増やして行きたいと考えている. それに向けた課題も明らかとなった.

1　目的

私は、SNS[*1]中毒者である[*2]．その症状として，Twitter[*3]を常に見ていないと落ち着かない[*4]・Twitter を開き，閉じようと思って閉じても気づくと Twitter を開き直している・Twitter のクライアントアプリを端末から削除しても，ブラウザからアクセスしてしまう[*4]・常に TweetDeck が画面に表示されており，作業に集中できない・Twitter をなかなか辞められないことに対する罪悪感を感じているも，結局やめることができない[*4]といったものがある．

また，世の中には SNS 中毒を含むインターネット中毒と思われる人が多数存在する．インターネット中毒の有病率は，イタリアの 0.8% から香港の 26.7% と調査によってばらつきがある[*5]ものの，世界中で一定数の人々が発症している [3]．

本課題では，私を含む多くの人々がインターネット中毒に悩んでいる状況を踏まえ，中毒状態から脱することをサポートするため，インターネット利用をコントロールするプロキシサーバを開発する．

2　概要

インターネット依存を断ち切るために，まずは利用時間の把握をすることが必要となる．「数分だけ使うつもりだったのに，ついつい時間が経っていた」現象を防ぐため，まずは客観的にオンラインでの活動時間を記録することが効果的とされている [4]．モニタリングによって時間についての問題意識を獲得したら，実際に利用時間を減らしていく必要がある．ここで有効な手段として，「外的ストップテクニック」と呼ばれる，一定時間にアラームが鳴ったらオンライン活動をストップするというものがある [4]．これにより，少しづつ時間のコントロールを取り戻していくのである．

そこで，WEB サービスにアクセスしてから一定時間後に通信を遮断し，また一定時間が経過したら通信を再開する HTTP プロキシがあれば，時間感覚を取り戻しながら，ネットの利用頻度を減らすことができると考え，実装した．プログラムの名前を detox-proxy とした．

図 1　作成したロゴ
スイッチの記号 (JIS C 0617) をアレンジした．

3　HTTP(S) プロキシの仕組み

このセクションでは，HTTP/1.1 における一般的な HTTP プロキシの動作原理について，簡単に触れておく．

プロキシは，クライアントとサーバの通信の間に挟まり，通信を仲介するものとして捉えるとわかりやすい．

3.1　HTTP プロキシ

まず，OS やブラウザの設定で HTTP プロキシを有効にしておく，または環境変数 `HTTP_PROXY` にプロキシサーバのアドレスを指定しておく．こうすると，自動的にアプリケーションの HTTP 通信がプロキシを経由するようになる．

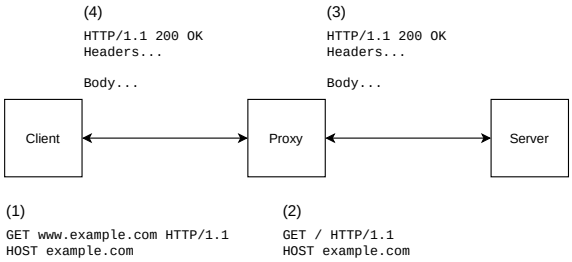

図 2　HTTP プロキシの動作

図 2 に，HTTP プロキシの動作を図解した．まず，クライアントがプロキシに向けて，(1) のリクエストを送る．ここで，メソッドの隣のパスの形式は，通常のリクエストと違い，絶対パスとなっている．また，`HOST` ヘッダに，リクエスト先のサーバを指定する．

次に，リクエストを受け取ったプロキシは，`HOST` ヘッダに記されているアドレス宛に，(2) のリクエストを送る．ここで，基本的にクライアントが付けたヘッダは改変せず，そのまま送信する[*6]．body が含まれる場合は，そちらも改変せずに送信する．

リクエストを受け取ったサーバは，通常のクライアントからのアクセスと同じように[*7]，(3) のレスポンスを返す．レ

[*1] Social Networking Service．WEB 上にて，社会的つながりを構築し，コミュニケーションを取るプラットフォーム．代表的なサービスに，Facebook, Twitter などがある．

[*2] 河井ら [1] の SNS 依存尺度に基づく．

[*3] 米 Twitter 社が運営する，短文を中心とした SNS サービス．`https://twitter.com`

[*4] 順に，離脱症状・再燃現象・葛藤 (Intrapersonal conflict) [2]

[*5] ばらつきがある理由として，全世界的な調査が不足していることや，ネット中毒の定義・基準，調査の方法論が確立していないことが挙げられる [3]．また，インターネットの利用環境は年々変化しており，これもばらつきの一つの要因になっていると考えられる．

[*6] Forwarded ヘッダ [5] など，必要に応じてプロキシが書き換えるケースもある．

[*7] プロキシからのアクセスなのか，通常のクライアントからのアクセスかを見極めるのは一般的に困難である．

スポンスを受け取ったプロキシは, その内容をそのままクライアントへと返す (4).

このような動作で, プロキシはクライアントとサーバとのやり取りを仲介するのである. (1) のリクエストを受け取った段階で, HOST ヘッダを見て通信を遮断するといったことや, (3) のレスポンスのヘッダやボディを見て有害なものが含まれていないかチェックするといったことが可能となる.

3.2 HTTPS プロキシ

セクション 3.1 で紹介した HTTP プロキシの仕組みは, そのまま HTTPS 通信に用いることができない. なぜなら HTTPS 通信においては, 通信が暗号化されており, ヘッダなどの中身をプロキシが見ることはできないからである[*8]. サーバとクライアント間のエンドツーエンド暗号化を実現しつつ, プロキシサーバを機能させるために, HTTP/1.1 では CONNECT メソッドが用意されている [6].

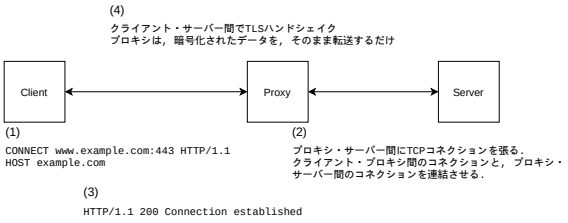

図 3　HTTPS プロキシの動作

図 3 に, CONNECT メソッドを用いたプロキシの動作を示した. まず, クライアントはプロキシに CONNECT メソッドを使ってリクエストを送信する. リクエストを受け取ったプロキシは, パスに記されているサーバのアドレス・ポートに TCP コネクションを張る. そして, これ以降クライアントから送られてきたデータは, 直接このコネクションへ流し込む. また, サーバから送られてきたデータも, そのままクライアントとのコネクションへ流す. これにより, 仮想的にクライアント・サーバ間に 1 本のコネクションが貼られたことになる (2).

その後, プロキシはクライアントへ 200 番台のレスポンスを返す (3). クライアントは, これでプロキシの先にサーバへのコネクションが張られたとみなし, 通常のプロキシを介さないアクセスと同じ手順で TLS ハンドシェイクを開始し, サーバとセキュアな通信を行う.

4　仕様と実装

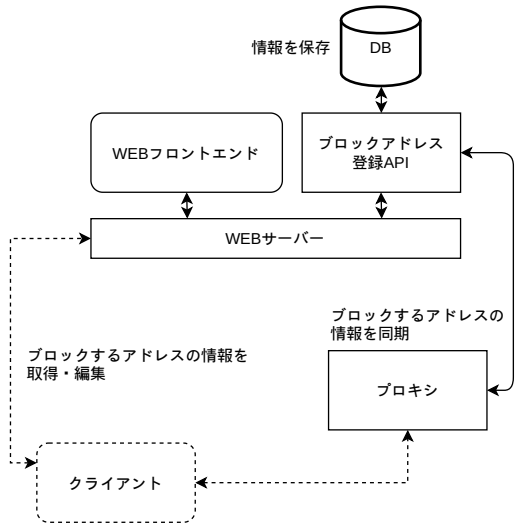

図 4　detox-proxy の全体像

図 4 に, detox-proxy の構成を示した. まず, ブロックするアドレス・遮断時間などを登録するデータベースと, それに対し CRUD 操作を行う REST API サーバを用意する. また WEB のフロントエンドを用意し, ユーザはフロントエンドを介して API にリクエストを送り, ブロックするアドレスを登録・編集できるようにする. プロキシは, API から登録されたブロックアドレスを取得し, 内部のメモリに保存する. クライアントからの HTTP リクエストが来るたびに, リクエスト先のアドレスがブロックリストに登録されていないか, されている場合制限時間内のアクセスかを把握し, 通信の仲介・遮断を行う.

なお, 開発を効率化するため, API・DB・WEB サーバ・プロキシはコンテナ化した. また, それらのコンテナをオーケストレーションツールである docker-compose を用いて管理し, 用意にデプロイ・起動できるようにした.

以下に, フロントエンド・API サーバ・プロキシサーバについてその仕様と, 実装を説明する.

4.1　フロントエンド

フロントエンドは, Elm[7] で実装した. Elm は強い静的型付けを持つ関数型の DSL [*9] で, Haskell[8] に似た文法を持つ. Elm は, Javascript に変換されるトランスパイル言語である.

[*8] なおプロキシサーバの証明書を発行し, 通信の途中 (プロキシ) で通信内容を復号し, チェックした上で再度暗号化して転送するものも存在する. 本稿では扱わない.

[*9] Domain-Specific Language. 汎用言語と異なり, 特定のタスクに特化した言語. Elm は, WEB フロントエンドの開発に特化した言語である.

[*10] プログラムを, Model・View・Update に分割し, それぞれにアプリケーションの状態を保持・状態を HTML に変換・メッセージを使って状態を更新する役割を担わせる. これにより, 見通しのよいプログラムが生まれる.[9]

[*11] Single Page Application の略.

TEA アーキテクチャ[*10] に沿って, 副作用のない安全な SPA[*11]を作ることができ, また変化の激しいフロントエンド界隈から一歩距離を置き, 穏やかに開発できることから, この言語を採用した. また, デザインは図 5 のように, Neumorphism[*12] を取り入れた, 立体感のあるわかりやすいものとした.

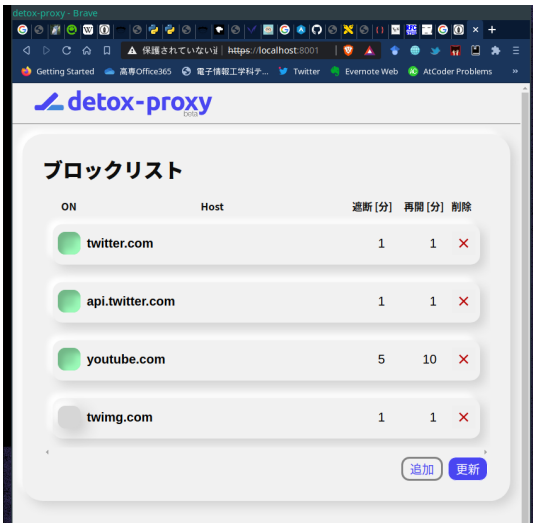

図 5　フロントのスクリーンショット

4.2　API サーバ

API サーバは, Python 及びマイクロ・WEB アプリケーション・フレームワークである FastAPI[10] を使用した. FastAPI は, 簡単に Python の型ヒントに基づいて, REST API が構築できる特徴を持つ.

作成したエンドポイントは, 次の 5 つである.

GET /api/blockaddress
概要　ブロックアドレスのリストを取得する.
Response　`List[Block]`

POST /api/blockaddress
概要　ブロックアドレスを新規登録する
Request　`List[BlockCreate]`
Response　`List[Block]`

PUT /api/blockaddress
概要　既存のブロックアドレスの情報を更新する.
Request　`List[Block]`
Response　`List[Block]`

DELETE /api/blockaddress
概要　登録されたブロックアドレスを削除する.
Request　`List[Int]` (Block の id を指定)

GET /proxy.pac
概要　PAC ファイル[*13]をダウンロードする.
Response　PAC ファイル.

なお, 上記中の `Block`・`BlockCreate` は, ソースコード 1 に示す形式の JSON オブジェクトである.

ソースコード 1　各 JSON オブジェクト

```
Block: {
  id : integer ブロックリストの ID
  url : string ブロックするアドレス
  start : integer 初回アクセスからｓｔａｒｔ分経過したら遮断
  end : integer 初回アクセスからｓｔａｒｔ+ｅｎｄ分経過したらリセット
  active : boolean このブロック情報が有効かどうか
}

BlockCreate: {
  url : string
  start : integer
  end : integer
  active : boolean
}
```

また, API サーバの起動時に, `GET /api/blockaddress` で取得できるものと同じデータをプロキシサーバに送信し, ブロックリストの同期を図る.

4.3　プロキシサーバ

detox-proxy の中核となるプロキシは, Nim[12] で実装した. Nim は, Python に似た文法を持つ, 静的型付け言語である. C 言語にトランスパイルされる[*14]ため, 高速なプログラムを実装することができるという特徴を持つ. また `async/await` 構文による非同期処理が書きやすく, ネットワークプログラミングに適していると考え採用した.

プロキシサーバでは, まず TCP ソケットを作成し, 5001 番ポートで listen する. クライアントからのアクセスがあれば, HTTP リクエストをパースし, リクエストされたサー

[*12]スキューモーフィズムとフラットデザインの両方を取り入れたようなデザイン.
[*13]Proxy Auto-Configuration. プロキシの設定情報を含むファイル. 内部には Javascript が書かれており, アクセス先に応じて使うプロキシを変更すると言った挙動も可能. [11]
[*14]C 言語以外にも, C++, Objective-C, Javascript など, いくつかの言語にトランスパイルすることができる.

バへ新たに TCP コネクションを張る．その後，リクエストの種類に応じてセクション 4.3.1 及び 4.3.2 に記した挙動をする．

4.3.1 HTTP リクエスト (CONNECT メソッド以外)

クライアントから送られたデータを，サーバへ流す．サーバからレスポンスを受け取ったら，そのままクライアントへ送信する．以上の動作が終了したら，クライアント・プロキシ間，プロキシ・サーバ間それぞれの TCP コネクションを切断する．

4.3.2 HTTPS リクエスト (CONNECT メソッド)

クライアントへ，`HTTP/1.1 200 Connection Established` レスポンスを返す．その後，クライアント・プロキシ間及び，プロキシ・サーバ間の TCP コネクションを結合させ，双方向にデータを転送する．2 つのコネクションを監視し，どちらかのコネクションの切断を検知したら，自動的にもう一方も切断する．

4.3.3 遮断処理

本セクションでは，指定されたアドレスとの接続を，遮断する処理について解説する．

API サーバの起動時に，ソースコード 1 中に示した `Block` データが送られてくるので，メモリ上に保持する．また，ブロックリストに含まれるアドレスについては，アクセスログもメモリ上に保持する．

クライアントからリクエストが来るたびに，アクセス先のアドレスがブロックリストに含まれていないかチェックし，含まれていなければ接続する．含まれていた場合は，アクセスログを参照し，アクセスから `Block.start` 分経過していなければ接続する．経過していた場合は，`Block.start + Block.end` 分経過しているかチェックし，経過していれば接続し，経過していなければ接続を遮断する (TCP コネクションを切断する)．

5 結果

求めていたプロキシが実装できた．実際に 1 日使用してみたところ，Twitter の利用頻度・ツイート数が減少し，締め切り直前のレポート執筆作業に集中することができた．

6 今後の課題と解決策

今後は，プロキシサービスを運用し，多くの人に使ってもらい，このプロキシがインターネット中毒の治療に効果を発揮するのか検証していきたいと考えている．それに向け，現在明らかになっている課題を以下に示す．

6.1 セキュアな接続

現在，クライアント・プロキシ間が平文でやり取りされている．これは，HTTP 接続だけでなく，HTTPS 接続において `CONNECT` リクエストをプロキシに送信するときも同様である．このままでは，通信内容を途中で第 3 者に把握されたり，改ざんされる危険性がある．

そこで，クライアント・プロキシ間の通信を TLS 化する必要がある．実際に実装したが，アプリケーションによって対応がまちまち[*15] で，安定した接続ができなかったため，コードを削除した．

6.2 認証

API サーバにユーザ登録機能を設け，ユーザごとにブロックするアドレスを設定できるようにする必要がある．またプロキシにはユーザ名とパスワードによる BASIC 認証機構を実装し，ユーザに応じた社団処理を行う必要がある．

こちらについても実装した．しかし，アプリケーションによって BASIC 認証の対応がまちまちであり，OS の設定で認証を有効化しても，ブラウザ以外の殆どのアプリケーションが `Proxy-Authorization` ヘッダを付与せず，また 407 レスポンスに対応しなかった．安定して動作しなかったため，削除した．

6.3 解決策

セクション 6.1・6.2 で示した課題を解決するため，TCP・UDP のプロキシを作成し，各クライアントに作成した TUN デバイスなどからアクセスすることでこうしたアプリケーションの制約を受けないようにし，想定通りのサービスを実現させたい．

7 終わりに

HTTP についての理解もあやふやだった自分が，時間をかけつつも一から HTTP(S) プロキシを実装できたことに，大変な喜びを感じている．長期間に渡る使用を通じて，SNS 依存から脱却し，生産的な学生生活が送れるようになることを期待している．

今後は，ネットワークプログラミングについて深く学び，セクション 6.3 で示した TCP・UDP プロキシの実装に励みたい．最終的にはプロキシサービスを運営し，世界中のインターネット依存症で苦しむ人のサポートをしたい．

[*15] cURL[13], Chrome[14], Firefox[15] といったアプリケーションは対応している．その一方で，ほとんどのアプリケーションは対応の兆しを見せておらず，プロキシサーバが 426 を返しても無反応のまま通信が途切れてしまう．

参考文献

[1] 河井大介, 天野美穂子, 小笠原盛浩, 橋元良明, 小室広佐子, 大野志郎, and 堀川裕介, "Sns 依存と sns 利用実態とその影響," in 日本社会情報学会全国大会研究発表論文集 日本社会情報学会 第 *26* 回全国大会, 日本社会情報学会, 2011, pp. 265–270.

[2] M. Griffiths, "A 'components' model of addiction within a biopsychosocial framework," *Journal of Substance Use*, vol. 10, no. 4, pp. 191–197, 2005. DOI: 10.1080/14659890500114359. eprint: https://doi.org/10.1080/14659890500114359. [Online]. Available: [https://doi.org/10.1080/14659890500114359%20](https://doi.org/10.1080/14659890500114359).

[3] D. J Kuss, M. D Griffiths, L. Karila, and J. Billieux, "Internet addiction: A systematic review of epidemiological research for the last decade," *Current pharmaceutical design*, vol. 20, no. 25, pp. 4026–4052, 2014.

[4] エルサルヒ・ムハンマド, 村松太郎, 樋口進, and 三村將, "インターネット依存の概念と治療 (特集 アディクション：行動の嗜癖)," *Brain and nerve*, vol. 68, no. 10, pp. 1159–1166, Oct. 2016, ISSN: 1881-6096 (v. 59, no. 1-). [Online]. Available: https://ci.nii.ac.jp/naid/40020973164/.

[5] A. Petersson and M. Nilsson, *Rfc 7239 - forwarded http extension*, https://tools.ietf.org/html/rfc7239, Jun. 2014.

[6] E. R. Fielding and E. J. Reschke, *Rfc 7231 - hypertext transfer protocol (http/1.1): Semantics and content*, https://tools.ietf.org/html/rfc7231, Jun. 2014.

[7] E. Czaplicki, *Elm - delightful language for reliable web applications*, https://elm-lang.org/, (Accessed on 03/04/2021).

[8] haskell.org, *Haskell language*, https://www.haskell.org/, (Accessed on 03/04/2021).

[9] *The elm architecture · an introduction to elm*, https://guide.elm-lang.org/architecture/, (Accessed on 03/04/2021).

[10] *Fastapi fastapi framework, high performance, easy to learn, fast to code, ready for production*, https://fastapi.tiangolo.com/, (Accessed on 03/04/2021).

[11] MDN contributors, プロキシ自動設定ファイル - *http — mdn*, https://developer.mozilla.org/ja/docs/Web/HTTP/Proxy_servers_and_tunneling/Proxy_Auto-Configuration_PAC_file, (Accessed on 03/05/2021).

[12] *Nim programming language*, https://nim-lang.org/, (Accessed on 03/04/2021).

[13] *Https proxy with curl — daniel.haxx.se*, https://daniel.haxx.se/blog/2016/11/26/https-proxy-with-curl/, (Accessed on 03/04/2021), Nov. 2016.

[14] *Secure web proxy - the chromium projects*, https://dev.chromium.org/developers/design-documents/secure-web-proxy, (Accessed on 03/04/2021).

[15] *378637 - add support for connecting to http proxy over https*, https://bugzilla.mozilla.org/show_bug.cgi?id=378637, (Accessed on 03/04/2021), May 2014.

[16] kekeho(Hiroki Takemura), *License*, https://github.com/kekeho/detox-proxy/blob/master/LICENSE.

ソースコード

本課題で作成したソースコードは, https://github.com/kekeho/detox-proxy にて, MIT ライセンス [16] で提供している.

備考

本課題の他に, 環境情報学部入学予定 後藤 大介君と一緒に, 2021 年入学の SFC 生が現在行っているプロジェクトを共有するページ (https://projshare.com/) の実装を行った. 主にサーバサイドを担当した.