Uniwersytet Warszawski
Wydział Matematyki, Informatyki i Mechaniki

Łukasz Jancewicz
Nr albumu: 209248

Anonimowa publikacja treści w Internecie

Praca magisterska
na kierunku INFORMATYKA

Praca wykonana pod kierunkiem
dr Janiny Mincer-Daszkiewicz

Maj 2007

Oświadczenie kierującego pracą

Potwierdzam, że niniejsza praca została przygotowana pod moim kierunkiem i kwalifikuje się do przedstawienia jej w postępowaniu o nadanie tytułu zawodowego.

Data Podpis kierującego pracą

Oświadczenie autora pracy

Świadom odpowiedzialności prawnej oświadczam, że niniejsza praca dyplomowa została napisana przeze mnie samodzielnie i nie zawiera treści uzyskanych w sposób niezgodny z obowiązującymi przepisami.

Oświadczam również, że przedstawiona praca nie była wcześniej przedmiotem procedur związanych z uzyskaniem tytułu zawodowego w wyższej uczelni.

Oświadczam ponadto, że niniejsza wersja pracy jest identyczna z załączoną wersją elektroniczną.

Data Podpis autora pracy

Streszczenie

Publikacja treści w Internecie zwykle wiąże się z ryzykiem ujawnienia tożsamości publikującego, co może doprowadzić do przykrych dla niego konsekwencji. Istnieją aplikacje zapewniające całkowitą anonimowość w sieci, ale ich działanie jest bardzo powolne, a używanie niewygodne. Z kolei aplikacje służące do wymiany plików są nieprzystosowane do zapewniania anonimowości. W pracy przedstawiono projekt i realizację sieci *Sandstorm*, która zapewnia częściową anonimowość – dla nadawców, nie dla odbiorców – oraz aplikacji służącej do wymiany plików, korzystającej z tej sieci. Przeprowadzono testy wydajności aplikacji i sieci. Opis realizacji projektu został poprzedzony analizą sposobów zapewniania anonimowości, porównaniem istniejących aplikacji zapewniających anonimowość oraz opisem sposobów ustalania tożsamości użytkownika w sieci.

Słowa kluczowe

anonimowość, sieci punkt-do-punktu, sieci komputerowe, Internet, protokoły komunikacji, Python

Dziedzina pracy (kody wg programu Socrates-Erasmus)

11.3 Informatyka

Klasyfikacja tematyczna

C.2.1 Network Architecture and Design
C.2.2 Network Protocols
C.2.4 Distributed Systems
D.1.3 Concurrent Programming
H.3.3 Information Search and Retrieval

Tytuł pracy w języku angielskim

Anonymous Content Publishing on the Internet

Spis treści

Wprowadzenie

Podstawowym celem tworzenia sieci komputerowych jest umożliwienie wymiany informacji między użytkownikami pracującymi na różnych komputerach. Najbardziej spektakularnym przykładem sieci komputerowej jest oczywiście Internet, umożliwiający komunikację z komputerami właściwie we wszystkich częściach świata.

Internet jako medium komunikacyjne rozpoczął nową erę wymiany informacji – oto każdy mógł podłączyć się do sieci i wyrazić swoją opinię, napisać list lub umieścić zdjęcie czy film, i nikt nie żądał od niego uprawnień dziennikarskich. Nikogo nie interesowało, kim dana osoba jest, a jedynie co ma do powiedzenia czy przedstawienia w danej sprawie. Ta rewolucja w myśleniu spowodowała bardzo pozytywne efekty – ludzie zaczęli chętniej wymieniać się informacjami, poradami, opiniami i spostrzeżeniami. Czynnikiem, który umożliwiał taką swobodną wymianę myśli, była całkowita anonimowość – większość użytkowników Internetu używała pseudonimów i ukrywała się za niewiele mówiącymi adresami e-mail. Osoby mające do tej pory problemy z komunikacją międzyludzką mogły stworzyć sobie całkowicie nową tożsamość, zupełnie niezwiązaną z tą w świecie rzeczywistym, i uzyskać aprobatę społeczności internautów nie ze względu na to, kim byli, ale ze względu na to, co sobą reprezentowali.

Niestety, szybko zaczęły pojawiać się osoby i organizacje, którym owa anonimowość ułatwiała prowadzenie nielegalnej działalności. Do walki z internetową przestępczością zaangażowano siły policji w wielu krajach. Równolegle powstały ustawy, które nakładały na dostawców Internetu obowiązek monitorowania i przechowywania informacji o użytkownikach korzystających z ich usług. Szybko okazało się, że Internet nie jest tak anonimowy, jak na początku sądzono. Większość dzisiejszych sieci komputerowych, w tym Internet, bazuje na protokołach sieciowych zaprojektowanych na początku lat osiemdziesiątych zeszłego stulecia, takich jak *IP*, *TCP* i *UDP*. Protokół *IP*, aby dostarczyć wiadomość od jednego użytkownika do drugiego, posługuje się specjalnymi adresami, zwanymi *adresami IP*. Każda informacja przesyłana przez protokół *IP* zawiera adres nadawcy i odbiorcy. Posiadając taki adres i mając dane od dostawców internetu, można zwykle z łatwością ustalić adres fizyczny mieszkania czy budynku, z którego wiadomość została wysłana.

Wprowadzenie nadzoru Internetu znacząco zmniejszyło prowadzoną w nim nielegalną działalność, ale szybko zaczęło być stosowane do innych celów. Monitorowaniem Internetu zainteresowały się rządowe agencje cenzury, a także firmy, które uważały, że niektórzy użytkownicy sieci działają na ich szkodę. Rozpoczęły się aresztowania, pozwy i procesy użytkowników Internetu. Wolność słowa i wymiany informacji, dotąd niczym nieograniczona, została poważnie stłumiona. Jednak internauci, przyzwyczajeni już do swobody, jaką do tej pory dawał im Internet, zaczęli szukać rozwiązań, które pozwoliłyby im nadal zachować anonimowość.

Stworzono pierwsze implementacje tzw. „sieci w sieci", które miały zapewniać anonimowość w nieanonimowym środowisku Internetu. Używanie takiej sieci anonimowej zwykle polegało na zainstalowaniu specjalnej aplikacji, która komunikowała się z innymi aplikacjami tego samego rodzaju, wymieniając informacje w taki sposób, aby było bardzo trudno namierzyć miejsce (adres IP) ich pochodzenia oraz dostarczenia. Wymyślono wiele metod, jakie

stosowały i nadal stosują sieci anonimowe do zapewnienia anonimowości. Przede wszystkim w takich sieciach każdy użytkownik otrzymuje nowy adres, zupełnie niezwiązany z jego adresem IP. Następnie wiadomość jest adresowana tym nowym adresem i przekazywana przez kilku pośredników, z których żaden nie wie, w którym miejscu łańcucha przesyłania wiadomości się znajduje. Aby dodatkowo użytkownicy nie wiedzieli co przesyłają inni, korzystając z ich pośrednictwa, wiadomości są szyfrowane kluczami symetrycznymi i asymetrycznymi. Aby zwiększyć stopień anonimowości, szyfry można na siebie nakładać.

Sieci anonimowe w Internecie istnieją i funkcjonują, ale wciąż borykają się z wieloma problemami. Konieczność instalowania dodatkowej aplikacji powoduje, że treści w nich publikowane są dostępne dla dużo mniejszej rzeszy użytkowników, niż „zwyczajne" strony WWW, do oglądania których wystarczy przeglądarka internetowa. Dodatkowo wymiana informacji w takich sieciach przebiega bardzo powoli, ze względu na konieczność przesłania wiadomości przez pośredników, zaszyfrowania jej, znalezienia odbiorcy itp. Wciąż prowadzone są badania mające na celu zwiększenie prędkości sieci anonimowych przy zachowaniu ich podstawowej zalety – anonimowości.

Celem niniejszej pracy była analiza działania sieci anonimowych w Internecie z punktu widzenia potrzeb bardzo specyficznego użytkownika, któremu nie zależy na pełnej anonimowości, a jedynie na anonimowości osób udostępniających treści. Taka połowiczna anonimowość w zupełności wystarcza osobom, które chcą uniknąć cenzury lub które po prostu, przekazując jakąś informację, nie chcą ujawniać swojej tożsamości. Nie zapewnia natomiast bezpieczeństwa przestępcom, gdyż adres odbiorcy informacji nie jest ukryty i można go wyśledzić w taki sam sposób, jak w sieci nieanonimowej.

Dzięki zmniejszeniu wymagań anonimowości o połowę, możliwe jest zwiększenie wydajności sieci anonimowych. Mniejsza liczba komputerów, przez które musi przechodzić każda wiadomość, szybsze wyszukiwanie informacji – to tylko niektóre z zysków, jakie osiągamy. W ramach niniejszej pracy powstała koncepcja nowej sieci, zapewniającej anonimową publikację treści w Internecie. Zasada działania sieci została oparta na rozwiązaniach stosowanych z sukcesem przez popularne sieci zapewniające pełną anonimowość, a także na koncepcjach autora, których realizacja przyspiesza działanie sieci w porównaniu do sieci w pełni anonimowych.

Sieć została stworzona na bazie rozwiązań stosowanych z sukcesem przez popularne sieci zapewniające pełną anonimowość, a także na koncepcjach autora, które przyspieszają jej działanie w porównaniu do sieci w pełni anonimowych.

Integralną częścią pracy jest implementacja zaprojektowanej sieci, nazwana *Sandstorm* (od burzy piaskowej, w której nic nie widać), wraz z aplikacją służącą do wymiany plików. Aplikacja ta pozwala w wygodny sposób udostępniać i pobierać pliki z zachowaniem anonimowości osób udostępniających.

Struktura pracy

Dalsza część pracy zorganizowana jest w następujący sposób:

Rozdział 1 zawiera definicje podstawowych pojęć, takich jak węzeł, sieć punkt-do-punktu czy anonimowość. Pojęcia te są używane w dalszych rozdziałach pracy. W rozdziale tym zaprezentowano również sposoby poznania tożsamości użytkownika w sieci oraz sposoby zachowania anonimowości.

W rozdziale 2 zamieszczono przegląd istniejących sieci zapewniających anonimowość, wymieniono ich charakterystyczne cechy, zalety i wady. Przedstawiono w nim również sieci nieanonimowe stworzone specjalnie do szybkiej wymiany plików. Kompilacja wniosków z prze-

glądu obu rodzajów sieci doprowadziła do stworzenia założeń projektu nowej sieci, zapewniającej anonimową publikację treści.

W rozdziale 3 opisano, jak przebiegało przygotowywanie i tworzenie sieci zapewniającej anonimową publikację treści, *Sandstorm*, oraz aplikacji bazującej na tej sieci. Opisano wybrane do implementacji: język programowania i biblioteki. Opisana została funkcjonalność sieci *Sandstorm*, wraz z przykładami działania. Następnie przedstawiono zasadę działania sieci, algorytmy i protokoły, z których korzysta oraz architekturę aplikacji – podział na pakiety, moduły i wątki. Rozdział zawiera również dokładną analizę anonimowości, jaką zapewnia stworzona sieć, wraz z listą możliwych ataków na anonimowość i sposobami ich zapobiegania.

Rozdział 4 zawiera opis przeprowadzenia testów wydajności sieci *Sandstorm* oraz wyniki tych testów.

W rozdziale 5 zostały przedstawione możliwości dalszej pracy nad rozwojem stworzonej sieci. Zamieszczono w nim również podsumowanie pracy.

W dodatku A przedstawiono wymagania, proces instalacji oraz krótką instrukcję użytkownika aplikacji *Sandstorm*. Zawiera on również spis plików znajdujących się na płycie CD, dołączonej do niniejszej pracy.

Rozdział 1

Funkcjonowanie sieci punkt-do-punktu

W tym rozdziale zdefiniujemy pojęcia używane w dalszej części pracy oraz przedstawimy podstawowe zagadnienia związane z sieciami typu punkt-do-punktu.

1.1. Protokoły internetowe

Zacznijmy od zdefiniowania najbardziej podstawowych pojęć związanych z komunikacją między użytkownikami w sieci. Komunikacja w sieciach opiera się na *protokołach komunikacyjnych*.

Definicja 1.1. *W kontekście sieci komputerowych* protokoły komunikacyjne *to zbiór ścisłych reguł i kroków postępowania, które są automatycznie wykonywane przez urządzenia komunikacyjne w celu nawiązania łączności i wymiany danych.*

W sieciach komputerowych (do których zaliczamy również Internet) protokoły nakładają się na siebie, tworząc tzw. *warstwy protokołów*. Najpopularniejszym protokołem warstwy 3. (zwanej także warstwą sieciową), jest protokół *IP* (*Internet Protocol*, [16, 15]), stosowany m.in. w Internecie.

W protokole *IP* dane są dzielone na *pakiety*, czyli fragmenty o ustalonej strukturze. Każdy pakiet składa się z nagłówka i treści.

Rysunek 1.1 przedstawia schemat nagłówka protokołu *IP*.

<table>
<tr><td>Bity 0-3</td><td>4-7</td><td>8-15</td><td>16-18</td><td>19-23</td><td>24-31</td></tr>
<tr><td>Wersja</td><td>IHL</td><td>Typ usługi</td><td colspan="3">Długość całkowita</td></tr>
<tr><td colspan="3">Identyfikator</td><td>Flagi</td><td colspan="2">Pozycja fragmentu</td></tr>
<tr><td colspan="2">Czas życia</td><td>Protokół</td><td colspan="3">Suma kontrolna nagłówka</td></tr>
<tr><td colspan="6">Adres źródłowy</td></tr>
<tr><td colspan="6">Adres docelowy</td></tr>
<tr><td colspan="6">Opcje</td></tr>
</table>

Rysunek 1.1: Nagłówek protokołu IPv4

Najbardziej interesujące z punktu widzenia zapewniania anonimowości są dwa pola: adres źródłowy oraz adres docelowy. Dzięki tym polom rutery w sieci Internet wiedzą, do kogo przesłać wiadomość oraz do kogo przesłać odpowiedź na nią.

Dwoma najpopularniejszymi protokołami warstwy 4. (czyli tej bezpośrednio powyżej protokołu *IP*) są protokoły: *TCP* oraz *UDP*.

Protokół *TCP (ang. Transmission Control Protocol)* [18] został zaprojektowany z myślą o stworzeniu niezawodnych połączeń w środowisku zawodnych protokołów niższych warstw. Zapewnia on wyższym warstwom sieciowym wrażenie niezawodnej komunikacji przez system tzw. *gniazd (ang. sockets)*. W protokole *TCP* na każdą wiadomość przysłaną z komputera A do komputera B, komputer B odpowiada komputerowi A potwierdzeniem, że wiadomość dostał w nienaruszonym stanie. Jeśli komputer A nie otrzyma potwierdzenia, to próbuje wiadomość przesłać ponownie.

Drugi z wymienionych protokołów, *UDP (ang. User Datagram Protocol)* [20], nie zapewnia niezawodności, jego zaletą jest natomiast szybkość dostarczania pakietów – nie są one obarczone koniecznością potwierdzania odebrania pakietu nadawcy. Z tego powodu z protokołu *UDP* korzysta się w Internecie tam, gdzie szybkość jest ważniejsza niż spójność danych, np. przy transmisjach audio czy wideo.

Zauważmy, że skoro w protokole *UDP* nie jest potrzebna odpowiedź do nadawcy, to w nagłówku pakietu *IP* nie korzysta się z pola „adres źródłowy" – to spostrzeżenie będzie istotne przy rozważaniu metod zapewniania anonimowości.

O protokołach IP, TCP i UDP oraz o warstwach protokołów sieciowych można przeczytać w [15].

1.2. Sieci punkt-do-punktu

Podstawową jednostką, która reprezentuje pojedynczego użytkownika sieci, będzie *węzeł*.

Definicja 1.2. Węzeł *(ang. peer, node) jest to pojedyncza aplikacja działająca na pewnym komputerze, podłączonym do sieci działającej na bazie protokołu TCP/IP (takiej jak Internet bądź lokalna sieć Ethernet).*

Grupa węzłów (punktów), która dąży do wymiany informacji, może stworzyć sieć, łącząc się między sobą – stąd *sieć punkt-do-punktu*.

Definicja 1.3. Sieć punkt-do-punktu *(ang. peer-to-peer network) jest to zbiór* węzłów *korzystających z tej samej aplikacji (ściślej: z tego samego* protokołu komunikacji punkt-do-punktu *działającego na bazie protokołu TCP/IP) i połączonych w pewien, niekoniecznie spójny, graf bezpośrednich połączeń. Zwykle węzły sieci punkt-do-punktu nie są połączone każdy z każdym; każdy węzeł łączy się jedynie z kilkoma* sąsiadami*, co w założeniu wystarcza do komunikowania się z dowolnym węzłem w sieci poprzez kilku pośredników.*

Jeśli węzeł jest połączony za pomocą pewnego protokołu komunikacji z co najmniej jednym węzłem w pewnej sieci punkt-do-punktu, to powiemy, że jest on *podłączony* do tej sieci punkt-do-punktu. Węzeł łączy się z resztą sieci punkt-do-punktu za pośrednictwem *sąsiadów*.

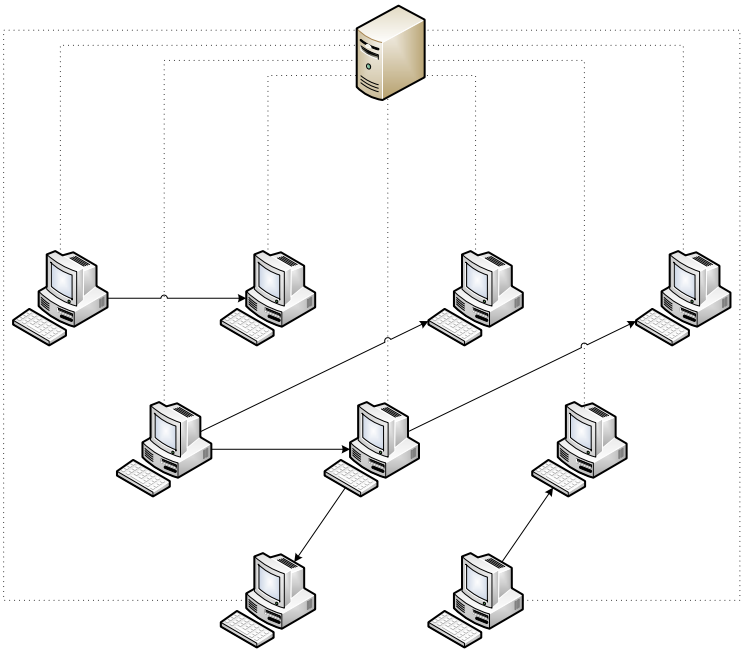
Definicja 1.4. Sąsiadem *węzła nazwiemy węzeł w sieci punkt-do-punktu, z którym jest on połączony stałym, bezpośrednim połączeniem protokołem TCP bądź UDP. W szczególności oznacza to, że węzeł zna adresy IP wszystkich swoich sąsiadów.*

Sieci punkt-do-punktu dzielimy na *scentralizowane* i *zdecentralizowane*.

Definicja 1.5. Scentralizowana sieć punkt-do-punktu *jest to sieć, której działanie jest uzależnione od pewnej stałej liczby węzłów, które muszą być obecne i działające, aby sieć mogła poprawnie funkcjonować.*

Takie superwęzły nazywane są *serwerami* i zwykle mają większe uprawnienia, niż pozostali użytkownicy sieci. Na przykład, w sieci *eDonkey2000* [25] serwery są odpowiedzialne za przechowywanie informacji o użytkownikach, o tym, jakie pliki każdy z podłączonych do nich użytkowników udostępnia, jaki jest jego adres IP itp. Każdy węzeł sieci musi być podłączony do któregoś z serwerów, aby mógł działać.

Rysunek 1.2 przedstawia przykładowy schemat scentralizowanej sieci punkt-do-punktu.



Rysunek 1.2: Przykładowa scentralizowana sieć punkt-do-punktu. Linie przerywane oznaczają połączenia z serwerem. Strzałki oznaczają wymianę treści.

Na schemacie wyraźnie widać wyróżniony serwer, do którego podłączone są wszystkie węzły. Do serwera wysyłane są wszystkie komunikaty kontrolne i zapytania o pliki. Jedyna komunikacja, jaka zachodzi między dwoma węzłami, to przesyłanie zawartości plików.

Zaletami scentralizowanych sieci punkt-do-punktu są:

- prostota – bardzo łatwo jest zaprojektować i zrealizować projekt sieci scentralizowanej, a protokoły komunikacji są nieskomplikowane, gdyż opierają się głównie na komunikacji klient-serwer-klient;
- szybkość – przy założeniu, że serwer nie stanie się wąskim gardłem systemu, wyszukiwanie treści w takiej sieci polega na wysłaniu prostego zapytania do serwera, jest więc szybkie; jeśli serwer odpowie, że pliku nie ma, to jesteśmy pewni, że naprawdę nikt w sieci go nie posiada.

Scentralizowane sieci posiadają jednak wiele wad, które dyskwalifikują je w zastosowaniach wymagających zapewniania anonimowości:

- wrażliwość na awarię serwera – awaria bądź wyłączenie serwera powoduje, że węzły nie mogą się komunikować i wyszukiwać treści;

- informacje przechowywane na serwerze – osoba mająca dostęp do informacji, które znajdują się na serwerze, będzie potrafiła ustalić tożsamość oraz listę posiadanych plików każdego węzła.

Do naszych zastosowań dużo lepiej nadadzą się sieci zdecentralizowane.

Definicja 1.6. Zdecentralizowana sieć punkt-do-punktu *jest to sieć, w której wszystkie węzły mają te same uprawnienia i każdy z nich jest odpowiedzialny za utrzymywanie sieci.*

Zdecentralizowane sieci są tak zaprojektowane, by były odporne na odłączenie od sieci bądź awarię dowolnego węzła. Do podstawowych zalet zdecentralizowanych sieci punkt-do-punktu należą:

- niezależność – istnienie i działanie sieci nie jest uzależnione od żadnego konkretnego węzła;

- odporność na awarie – algorytmy protokołu sieciowego potrafią prawidłowo zareagować na awarię dowolnego węzła;

- rozproszenie informacji – rozłożenie informacji na poszczególne węzły może być tak zaprojektowane, by informacje z dowolnego węzła zawierały jedynie skrawek informacji o całej sieci.

Zdecentralizowane sieci punkt-do-punktu mają również swoje wady:

- podatność na rozspójnienie – awaria niektórych węzłów może spowodować, że spójna do tej pory sieć podzieli się na kilka mniejszych i węzły z jednej sieci nie będą już w stanie komunikować się z węzłami z innych sieci; aby temu zapobiec, każdy węzeł stara się utrzymać pewną liczbę połączeń z sąsiadami;

- powolność wyszukiwania – brak centralnego serwera powoduje, że każde wyszukiwanie musi przejść przez poszczególne węzły sieci – czas odpowiedzi jest więc dłuższy.

Rysunek 1.3 przedstawia przykładową, zdecentralizowaną sieć punkt-do-punktu.

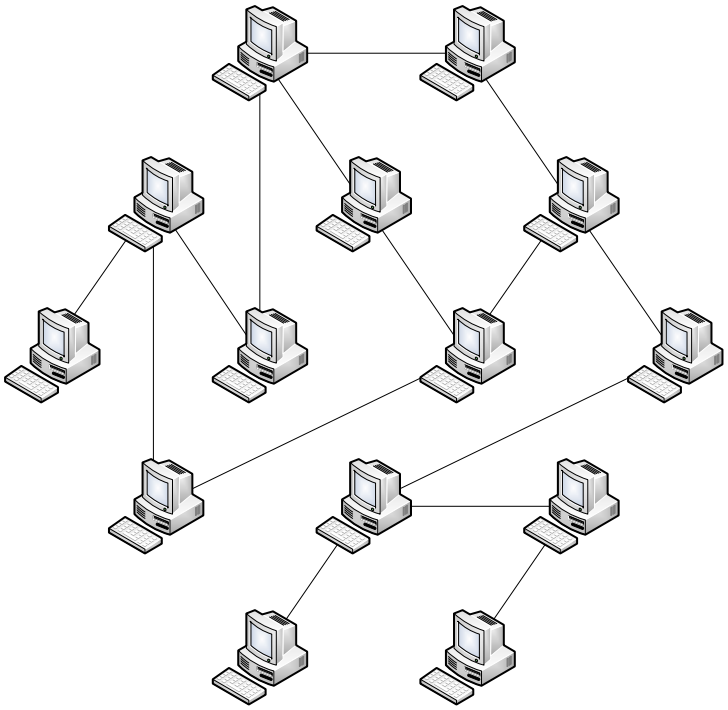
Widać, że w zdecentralizowanej sieci nie ma wyróżnionych węzłów – każdy z nich ma te same uprawnienia. Każdy węzeł posiada niewielki podzbiór węzłów całej sieci, z którymi jest połączony bezpośrednim połączeniem TCP/IP – są to sąsiedzi węzła. Komunikacja z pozostałymi węzłami jest nawiązywana jedynie przez pośredników.

Przykład dobrze pokazuje podatność zdecentralizowanych sieci na rozspójnienie: gdyby skrajnie prawy komputer został odłączony od sieci, podzieliłaby się ona na dwie mniejsze.

Celem powstawania sieci punkt-do-punktu jest przede wszystkim wymiana wszelkiego rodzaju *treści.*

Definicja 1.7. Treścią *(ang. content) nazwiemy informację przechowywaną w dowolnym formacie nadającym się do przesłania przez sieć.*

Tak więc treścią będą głównie pliki, ale też np. strumienie danych, takie jak transmisje audio czy wideo.



Rysunek 1.3: Przykładowa zdecentralizowana sieć punkt-do-punktu. Połączenia między węzłami oznaczają relację sąsiedztwa.

1.3. Ustalanie tożsamości użytkownika w sieci

Zidentyfikować użytkownika w sieci można na różne sposoby: wnioskując z jego wypowiedzi, przeszukując fora dyskusyjne i strony społecznościowe itd., ale najlepszym i najpewniejszym sposobem jest poznanie adresu źródłowego IP, który jest przesyłany w nagłówkach pakietów IP.

W zależności od dostawcy Internetu, użytkownik może mieć przydzieloną jedną z następujących kategorii adresów IP:

- stały adres IP – to znaczy taki, który nie zmienia się po rozłączeniu i ponownym podłączeniu do sieci;
- dynamiczny adres IP – przydzielany z pewnej puli losowo za każdym razem, gdy użytkownik rozłącza się i ponownie podłącza się do sieci;
- adres za wewnętrznym ruterem – do Internetu podłączony jest ruter, do którego jest z kolei podłączonych wielu użytkowników; każdy użytkownik jest z zewnątrz widziany jako ruter i ma adres IP rutera.

Dostawcy Internetu przechowują informację o tym, jakie adresy są komu przydzielone (zarówno statyczne, jak i dynamiczne), więc mając czyjś adres IP oraz chwilę jego uzyskania, można (jeśli ma się odpowiednie uprawnienia) dowiedzieć się od dostawcy, który jego klient korzystał w danej chwili z tego adresu. Cecha ta dotyczy dwóch pierwszych rodzajów adresów IP: stałego i dynamicznego. Trochę lepiej jest, jeśli jesteśmy za wewnętrznym ruterem – dużo rzadziej przechowywane są informacje o tym, kto w danej chwili korzystał z rutera.

Z tematem identyfikacji użytkownika mocno powiązane jest pojęcie tzw. *wiarygodnej zaprzeczalności*.

Definicja 1.8. *W sieciach komputerowych,* wiarygodna zaprzeczalność *(ang. plausible deniability) oznacza sytuację, w której użytkownik może zaprzeczyć, że to on jest autorem danej informacji, nawet, gdy została ona wysłana z jego komputera.*

W praktyce oznacza to, że użytkownikowi nie da się udowodnić, że jest on nadawcą danej informacji, gdyż równie dobrze jej nadawcą mógł być ktoś inny, a wiadomość tylko przechodziła przez komputer użytkownika.

1.4. Anonimowość w sieci

Zdefiniujmy dokładnie, co będziemy rozumieli przez anonimowość.

Definicja 1.9. *Powiemy, że pewna czynność (taka jak komunikacja, przesłanie treści itp.) w sieci punkt-do-punktu została dokonana* anonimowo*, jeśli żaden uczestnik tej czynności (poza wykonawcą) ani bierny obserwator nie potrafi jednoznacznie zidentyfikować wykonawcy czynności.*

W szczególności wystarczy nam jeśli wiadomo, że wykonawcą była osoba z pewnej grupy, ale nie wiadomo która, lub jeśli wiemy, że wykonawcą jest pewien użytkownik z pewnym statystycznym prawdopodobieństwem, które jest mocno mniejsze od 1 *(wiarygodna zaprzeczalność).*

Na przykład, wysłanie pewnej treści do innego węzła sieci będzie anonimowe, jeśli ten węzeł nie będzie potrafił ustalić, od kogo ją dostał. Z kolei odebranie treści będzie anonimowe, jeśli nadawca nie będzie wiedział, do kogo ją wysyła.

Pierwsze podejście do anonimowości wykorzystywało istniejące w strukturze Internetu serwery, zwane *serwerami pośredniczącymi.*

Definicja 1.10. Serwer pośredniczący *(ang. proxy server) jest to aplikacja uruchomiona na komputerze podłączonym do Internetu, która służy za pośrednika pomiędzy użytkownikami przeglądającymi strony WWW a serwerami, na których te strony się znajdują.*

Głównym celem powstawania serwerów pośredniczących była chęć zmniejszenia obciążenia serwerów oraz drogi, jaką musiały przebywać pakiety w sieci. Serwery takie uruchamiali bowiem najczęściej dostawcy internetu i pełniły one funkcję pamięci podręcznej stron WWW: za każdym razem, gdy użytkownik prosił o pobranie strony WWW, serwer pośredniczący sprawdzał, czy przed chwilą nie wysyłał tej samej strony innemu użytkownikowi i jeśli tak, to przesyłał jej kopię przechowywaną na twardym dysku zamiast przekazywać połączenia do serwera WWW ze stroną. Efektem ubocznym takich operacji była niemożność zorientowania się przez serwis WWW, kto (jaki adres IP) w danej chwili pobierał z niego treści.

Bardziej wyrafinowanym podejściem, z którego dla zapewnienia anonimowości często korzysta się w sieciach punkt-do-punktu, jest zastosowanie dodatkowej warstwy protokołu sieciowego, zwanej *siecią zaciemnioną.*

Definicja 1.11. Sieć zaciemniona *(ang. darknet) jest to sieć zbudowana na pewnym protokole warstwy 4. (zwykle TCP lub UDP). Sieć zaciemniona posiada własne protokoły komunikacji między węzłami, własne metody trasowania wiadomości oraz własne metody identyfikacji węzłów.*

Przypomnijmy definicję trasowania:

Definicja 1.12. Trasowanie *(ang. routing) to wyznaczanie trasy dla pakietu danych w sieci komputerowej, a następnie wysłanie go tą trasą.*

Oznacza to, że *sieć zaciemniona* tak naprawdę dubluje działanie protokołów IP oraz TCP/UDP, ale poprzez wprowadzenie własnego adresowania i nazywania węzłów umożliwia uniezależnienie się od ograniczeń protokołów IP oraz TCP i UDP. W szczególności *sieć zaciemniona* pozwala na zapewnianie anonimowości węzłów, mimo, iż ich adresy IP nadal są widoczne przez niektórych użytkowników.

Do identyfikacji węzłów w *sieciach zaciemnionych*, oraz do identyfikacji wielu innych obiektów, używa się tzw. *unikatowych identyfikatorów.*

Definicja 1.13. Unikatowy identyfikator *(często w skrócie:* UUID, Universally Unique Identifier*) jest to liczba bądź ciąg znaków, wygenerowany przez pewien generator pseudolosowy.* Unikatowy identyfikator *ma zwykle taką długość, że prawdopodobieństwo, iż dwukrotnie wylosowany zostanie ten sam identyfikator, jest pomijalne.*

Wiele własności sieci punkt-do-punktu opiera się na założeniu, że dwa niezależnie wygenerowane *unikatowe identyfikatory* będą różne.

Jeśli chcemy pobrać treść z sieci, to musimy potrafić ją jednoznacznie zidentyfikować. Do tego celu w Internecie stosuje się najczęściej *URI* [19].

Definicja 1.14. URI *(ang. Uniform Resource Identifier) jest standardem internetowym umożliwiającym łatwą identyfikację zasobów w sieci. URI jest zazwyczaj krótkim łańcuchem znaków, zapisanym zgodnie ze składnią określoną w standardzie. Łańcuch ten określa nazwę lub adres zasobu, który dany URI identyfikuje.*

Przykładami poprawnych *URI* są: adres strony WWW Uniwersytetu Warszawskiego `http://www.uw.edu.pl/` oraz identyfikator pliku pomocniczego dla programu *eMule* w sieci *eDonkey2000* `ed2k://|file|creator.zip|89922|135990DC0C1D792D426A7C005DDC8F92|/`. Należy zauważyć, że *URI* nie zawsze określa lokalizację zasobu (tym różni się od *URL, ang. Uniform Resource Locator*) – służy jedynie do jego jednoznacznej identyfikacji.

W przypadku plików najczęstszą stosowaną kombinacją jednoznacznie identyfikującą jest para: rozmiar pliku oraz jego *suma kontrolna.*

Definicja 1.15. Suma kontrolna *pliku to ciąg bajtów, który powstał przez zastosowanie pewnej funkcji skrótu na zawartości pliku. Funkcja skrótu jest dobierana w ten sposób, by zminimalizować prawdopodobieństwo, że dwa różne pliki otrzymają tę samą sumę kontrolną. Długość sumy kontrolnej nie przekracza zwykle 32 bajtów.*

Definicja 1.16. Funkcja skrótu *jest to funkcja, która przyporządkowuje dowolnie dużej liczbie będącej parametrem wejściowym (wiadomością) krótką wartość, zwykle posiadającą stały rozmiar, określaną jako skrót wiadomości.*

Najpopularniejszymi używanymi funkcjami skrótu są: MD4 [12], MD5 [13] i SHA-1 [17].

Anonimowość w *sieciach zaciemnionych* zapewnia się w ten sposób, że informacja po drodze od nadawcy do odbiorcy przechodzi przez kilka węzłów, które nazwiemy *pośrednikami.* W zależności od struktur, które wiążą pośredników, takie przekazywanie informacji daje różne stopnie anonimowości. Nas będzie interesował pewien szczególny typ struktury pośredników, zwany *tunelem.*

Definicja 1.17. Tunelem o długości N, którego właścicielem jest węzeł W*, nazwiemy uporządkowaną listę N węzłów sieci. Każdy z tych węzłów jest pośrednikiem i zna adresy IP poprzedniego i następnego pośrednika w tunelu. Poprzednikiem pierwszego pośrednika jest właściciel tunelu W, czyli węzeł, który będzie nadawał i odbierał wiadomości. Następnikiem ostatniego pośrednika jest wybrany przez właściciela adresat wiadomości.*

Tuneli używa się w następujący sposób:

- jeśli właściciel chce coś wysłać do odbiorcy – przekazuje adres odbiorcy i treść wiadomości pierwszemu pośrednikowi w tunelu; ten, jeśli jest ostatnim węzłem tunelu, wysyła wiadomość bezpośrednio do odbiorcy; jeśli zaś nie – przesyła adres odbiorcy i treść wiadomości kolejnemu pośrednikowi;

- jeśli odbiorca chce coś wysłać do właściciela – przekazuje treść wiadomości do węzła, od którego dostał bezpośrednio wiadomość tunelowaną od właściciela tunelu; wiadomość ta poprzez wszystkich pośredników dociera aż do właściciela.

Zauważmy, że sytuacja pierwszego pośrednika w tunelu jest taka sama jak każdego innego – w szczególności nie jest on świadomy, że jest pierwszym pośrednikiem i że łączy się bezpośrednio z właścicielem tunelu. Mamy więc tu idealny przypadek *wiarygodnej zaprzeczalności* dla właściciela tunelu – może on zawsze powiedzieć, że jest tylko kolejnym pośrednikiem. Używanie tuneli wiąże się jednak z dużą stratą wydajności – dla porcji danych o rozmiarze k bajtów, każdy pośrednik musi zużyć k bajtów swojego łącza odbierającego (przy odbieraniu danych od poprzednika) i k bajtów łącza wysyłającego (przy wysyłaniu do kolejnego pośrednika), co po dodaniu nadawcy i odbiorcy daje $(N+1)*k$ bajtów wszystkich łącz odbierających i tyle samo wysyłających. Oprócz utraty przepustowości sieci, tracimy również na czasie – czas przesłania wiadomości bezpośrednio od nadawcy do odbiorcy jest ok. $(N+1)$-krotnie niższy niż czas jej przesłania przez tunel.

Często oprócz zapewnienia anonimowości chcemy ukryć przed osobami postronnymi treść przesyłanych wiadomości. Do tych celów stosuje się szyfrowanie. Do przesyłania plików dobrze nadaje się tzw. *szyfrowanie symetryczne*.

Definicja 1.18. Szyfrowanie symetryczne *jest to sposób szyfrowania danych, który polega na tym, że używa się tego samego klucza do zaszyfrowania i rozszyfrowania wiadomości.*

Dzięki zastosowaniu tego samego klucza algorytm szyfrowania symetrycznego jest szybszy niż algorytm *szyfrowania asymetrycznego*, czyli takiego, w którym stosuje się inny klucz do szyfrowania, a inny do rozszyfrowania wiadomości. Jednym z najpopularniejszych algorytmów szyfrowania symetrycznego jest *Advanced Encryption Standard* (AES, [3]).

Jedynym problemem przy szyfrowaniu symetrycznym jest to, że należy zapewnić, aby nikt oprócz nadawcy i odbiorcy nie dowiedział się, jaki jest klucz. Jest kilka algorytmów, które potrafią doprowadzić do ustalenia klucza. Nazywają się one *protokołami uzgadniania klucza*.

Definicja 1.19. Protokół uzgadniania klucza *jest to schemat wymiany informacji pomiędzy dwoma użytkownikami, prowadzący do wygenerowania takiego samego klucza u obu użytkowników, a nieujawniający go osobom postronnym, które mogły podsłuchiwać całą konwersację.*

Najpopularniejszym protokołem uzgadniania klucza jest protokół Diffiego-Hellmana [4]. Przydatna będzie wiedza, na jakiej zasadzie funkcjonuje ten protokół. Załóżmy, że użytkownicy A i B chcą uzgodnić wspólny klucz do szyfrowania symetrycznego. Istnieje wiele wersji tego protokołu – przedstawiona tu wersja jest wykorzystywana w aplikacji dołączonej do niniejszej pracy.

1. Obaj użytkownicy znają liczby g oraz N. Aby algorytm był bezpieczny, liczby te muszą być odpowiednio dobrane, w szczególności N powinno być odpowiednio dużą (np. 1024-bitową) liczbą pierwszą. O dokładnych warunkach bezpieczeństwa protokołu Diffiego-Hellmana można przeczytać w [4, 14].

2. Użytkownik A losuje liczbę $p_a < N$. Następnie oblicza $t_a = g^{p_a} mod N$ i wysyła t_a do użytkownika B.

3. Użytkownik B losuje $p_b < N$, oblicza $t_b = g^{p_b} mod N$ i wysyła t_b do użytkownika A.

4. Użytkownik B otrzymuje t_a od użytkownika A. Oblicza $k = g^{p_a * p_b} mod N = {t_a}^{p_b} mod N$.

5. Użytkownik A otrzymuje t_b od użytkownika B. Oblicza $k = g^{p_a * p_b} mod N = {t_b}^{p_a} mod N$.

Jak widać, obaj użytkownicy obliczyli tę samą wartość klucza k bez konieczności przesyłania go przez sieć. Postronny obserwator nie byłby w stanie odtworzyć klucza k na podstawie liczb: g, N, t_a oraz t_b, które mógłby zdobyć.

Szyfrowania można użyć do utajnienia treści przesyłanych wiadomości na różnych etapach jej przesyłania. Nadawca może np. zaszyfrować wiadomość kluczem uzgodnionym z odbiorcą, a następnie ponownie zaszyfrować ją kluczem uzgodnionym z pierwszym pośrednikiem. Pierwszy pośrednik odszyfruje wiadomość kluczem uzgodnionym z nadawcą, następnie zaszyfruje kluczem uzgodnionym z kolejnym pośrednikiem i prześle dalej, itd.

Szczególnym typem szyfrowania, który ma związek z tunelami, jest tzw. *szyfrowanie czosnkowe*.

Definicja 1.20. Szyfrowanie czosnkowe *(zwane czasem również szyfrowaniem cebulowym) dla wiadomości, która ma dotrzeć do adresata A przez N pośredników polega na tym, by upewnić się, że każdy z pośredników przekazał wiadomość, a jednocześnie żaden z nich jej nie odczytał.*

Algorytm działania szyfrowania czosnkowego dla wiadomości W jest następujący:

1. dla każdego pośrednika p_i z $i \in 1..N$, ustal klucz szyfrujący k_i z tym pośrednikiem;

2. ustal klucz szyfrujący k_A z adresatem A;

3. zaszyfruj wiadomość W kluczem k_A, następnie kluczem k_N, potem kluczem k_{N-1} itd. aż do k_1;

4. wyślij zaszyfrowaną wiadomość do pośrednika p_1, który rozszyfruje ją kluczem k_1, następnie wyśle do pośrednika p_2, który rozszyfruje ją kluczem k_2 itd. aż do pośrednika p_N, który rozszyfruje ją kluczem k_N i prześle do adresata;

5. adresat po rozszyfrowaniu kluczem k_A otrzyma niezaszyfrowaną wiadomość W.

Zauważmy, że faktycznie wiadomość musi „przejść" przez wszystkich pośredników w tunelu w zdefiniowanej kolejności, aby mogła być poprawnie przeczytana przez adresata. Zapewnienie tej dodatkowej ochrony wiąże się jednak znowu z narzutem na wydajność – aby wysłać wiadomość przez tunel o długości N, nadawca musi wykonać $N + 1$ szyfrowań. Dla porównania, szyfrowanie samego tunelu i łącz między pośrednikami wymaga tylko jednego szyfrowania u nadawcy i u każdego pośrednika – następuje zrównoważenie obciążenia procesora na wszystkie węzły.

Aby szyfrowanie czosnkowe mogło funkcjonować poprawnie, potrzebny jest sposób uzgodnienia kluczy z każdym kolejnym pośrednikiem, przy czym musimy mieć pewność, że wiadomość nie zostanie przechwycona przez innego użytkownika (tzw. *atak man-in-the-middle*).

Rozdział 2

Projekt aplikacji

2.1. Zapotrzebowanie

Wśród treści dostępnych w Internecie jest pewna specyficzna kategoria, mianowicie takich, których posiadanie i pobieranie z Internetu jest bezpieczne, natomiast ich udostępnianie wiąże się z ryzykiem dla udostępniającego. Przykładami takich treści są:

- Artykuły, materiały filmowe, fotografie, nagrania przedstawiające pewne osoby lub zjawiska w niekorzystnym dla nich świetle. Autorzy tych materiałów mogą być narażeni na nieprzyjemności ze strony osób i organizacji, którym nie jest na rękę publikacja tych treści. Nikt natomiast nie będzie miał pretensji do osoby, która taki materiał obejrzy.
- Materiały, które mogłyby podlegać cenzurze, a ich autorzy represjom, tak jak dzieje się w wielu krajach o ustroju totalitarnym, np. w Chinach.
- Utwory chronione prawami autorskimi, które wolno posiadać na własny użytek. W polskim prawie są to: filmy, muzyka i książki po dacie ich oficjalnej publikacji. W interpretacji wielu polskich prawników, o ile pobieranie i przechowywanie tych utworów na własny użytek jest dozwolone, to ich rozpowszechnianie jest uznawane za nielegalne [21].

Wymiana plików z taką treścią powinna chronić osoby udostępniające, natomiast nie musi zapewniać anonimowości osobom pobierającym. Ponieważ każda warstwa zwiększająca anonimowość jednocześnie zmniejsza prędkość połączeń, przydatna byłaby aplikacja, która z jednej strony jest szybsza niż aplikacje zapewniające pełną anonimowość wszystkim użytkownikom, a z drugiej strony zapewnia anonimowość użytkownikom udostępniającym.

2.2. Istniejące aplikacje

W tym rozdziale przyjrzymy się istniejącym sieciom zapewniającym anonimowość. Następnie dokonamy przeglądu popularnych programów służących do wymiany treści i sprawdzimy je pod kątem przystosowania do anonimowości.

2.2.1. Sieci zapewniające anonimowość

Istniało wiele koncepcji sieci zapewniających mniejszy lub większy stopień anonimowości, ale zrealizowania i szerszego przyjęcia przez społeczność internautów doczekały się cztery z nich: *Freenet*, *Mute*, *I2P* oraz *Tor*.

Freenet

Sieć *Freenet* [1, 35] była jedną z pierwszych poważnych prób realizacji anonimowości w sieci. Na anonimowość we *Freenecie* wpływa kilka zastosowanych rozwiązań. Po pierwsze, każdy użytkownik *Freenetu* przeznacza część swojego dysku twardego na obsługę sieci. W losowych odstępach czasu, od losowych sąsiadów, pobierane są automatycznie losowe fragmenty plików, zupełnie niezwiązane z tym, jakich plików akurat szuka użytkownik. Owe fragmenty zapisywane są we wspomnianej części dysku twardego i znów w losowy sposób przesyłane do innych sąsiadów. Stworzony w ten sposób „szum informacji" utrudnia postronnemu obserwatorowi zorientowanie się, które pliki są tak naprawdę przeznaczone dla użytkownika.

Aby wyszukać plik w sieci, użytkownik musi znać jego *unikatowy identyfikator*, a następnie zapytać swoich sąsiadów, czy mogą mu go przesłać. Może się oczywiście zdarzyć, że odpowiedni plik już leży u użytkownika w losowo przechowywanych fragmentach. Sąsiedzi odpowiadają, jeśli plik mają, lub przesyłają zapytanie do swoich sąsiadów. Właściciel pliku nigdy nie łączy się bezpośrednio z użytkownikiem szukającym pliku, ale przesyła go przez wszystkich pośredników, przez których doszło do niego zapytanie. Tworzone przy wyszukiwaniu ścieżki przesyłu plików mogą zawierać do 16 węzłów pośrednich, co bardzo negatywnie odbija się na szybkości ich pobierania. Z tego powodu sieć *Freenet* jest uznawana za najwolniejszą, ale jednocześnie najbezpieczniejszą siecią anonimową.

Charakterystyczną cechą *Freenetu* jest sposób, w jaki nowy plik jest umieszczany w sieci. Otóż w odróżnieniu od wszystkich pozostałych sieci, umieszczenie nowego pliku nie polega na trzymaniu go na komputerze udostępniającego do czasu, aż kilku innych użytkowników go pobierze, ale udostępniający „wpycha" (ang. *push*) plik do sieci, wysyłając go swoim sąsiadom jako wspomniane losowe fragmenty. Następnie może bezpiecznie skasować udostępniany plik i nie będzie na jego komputerze śladu wskazującego na niego jako na udostępniającego.

Freenet nie udostępnia API dla aplikacji – jedynym programem do wymiany plików jest dołączany do dystrybucji program o tej samej nazwie.

Mute

Aplikacja i sieć *Mute* [29] (znów istnieje tylko jedna aplikacja korzystająca z tej sieci) opiera się na prostszym pomyśle niż *Freenet*. Użytkownik definiuje pulę plików, które zdecydował się udostępniać. Dla każdego pliku udostępnianego przez użytkowników tworzona jest *suma kontrolna* jego zawartości. Wyszukiwanie pliku polega na wysłaniu zapytania do sąsiadów o konkretną sumę kontrolną. Pytanie jest propagowane, aż dotrze do węzła, który posiada plik. Następnie, przy wykorzystaniu algorytmów mrówkowych [6], ustalana jest trasa przesyłu pliku do odbiorcy przez najmniejszą możliwą liczbę pośredników.

Sieć *Mute* jest popularna głównie wśród użytkowników zainteresowanych pobieraniem małych plików, ze względu na niską prędkość przesyłu.

I2P

Sieć *I2P* [27] prezentuje odmienną koncepcję w porównaniu do przedstawionych wcześniej. Sama w sobie nie służy do pobierania treści, a prezentuje jedynie interfejs dla aplikacji, które muszą być specjalnie napisane, aby mogły z niej korzystać. Idea anonimowości w *I2P* opiera się na koncepcji *tuneli wychodzących* oraz *tuneli przychodzących*. Każdy użytkownik sieci tworzy pewną liczbę tuneli wychodzących, przez które wysyła wszystkie informacje, oraz tuneli przychodzących, przez które pobiera wszystkie informacje. Każdy tunel składa się z pewnej liczby pośredników, którzy nie są świadomi swojej pozycji względem właściciela tunelu (inaczej mówiąc: nie wiedzą, dla którego użytkownika został stworzony tunel, którego są częścią).

Przesłanie dowolnej informacji odbywa się w następujący sposób:

- nadawca losuje tunel wychodzący i wysyła do pierwszego węzła w tunelu wiadomość oraz adres początku tunelu adresata;
- wiadomość przechodzi przez tunel wychodzący do ostatniego węzła, który wysyła ją na adres początku tunelu adresata;
- wiadomość przechodzi przez tunel przychodzący adresata do samego adresata.

Adresy w sieci *I2P* wyglądają jak zwykłe adresy internetowe, ale są zakończone nieistniejącą domeną `.i2p`. Najprostszym sposobem przeglądania jej treści jest więc używanie przeglądarki internetowej z odpowiednio skonfigurowanym serwerem pośredniczącym. Sam silnik aplikacji ma bogate możliwości konfiguracji, z możliwością ustalenia liczby i długości tuneli włącznie. Każdy przesyłany pakiet jest szyfrowany kolejno czterema algorytmami szyfrującymi: pomiędzy nadawcą a odbiorcą, pomiędzy pośrednikami w tunelu, pomiędzy tunelami oraz „czosnkowe" pomiędzy nadawcą a końcem tunelu.

Tor

Tor (*The Onion Router*, [5, 36]) jest to sieć anonimowa, która dodatkowo zapewnia połączenie ze „zwykłą", nieanonimową częścią Internetu. Rozwój sieci jest finansowany przez *Electronic Frontier Foundation*, organizację zajmującą się walką o wolność słowa w Internecie. W odróżnieniu od poprzednio wymienionych sieci, *Tor* dzieli węzły na dwie kategorie: serwery i klientów.

Serwery są to dedykowane maszyny rozmieszczone w różnych miejscach na świecie i podłączone do Internetu. Pełnią one funkcje zbliżone do ruterów sprzętowych w sieci Internet.

Klienci to aplikacje uruchamiane przez użytkowników. Klienci łączą się z Internetem lub innymi klientami poprzez tunel stworzony z pośredników, z których każdy jest serwerem. W ten sposób łącza użytkowników nie są zużywane do utrzymywania działania sieci, a ponieważ serwery mają zwykle sporą przepustowość, sieć Tor jest uznawana za najszybszą sieć anonimową.

W przypadku, gdy klient chce anonimowo połączyć się z nieanonimową stroną w Internecie, ostatni serwer-pośrednik w tunelu wysyła zapytanie do tej strony i przekazuje przez tunel odpowiedź – działa więc podobnie do serwera pośredniczącego.

2.2.2. Sieci wymiany plików

Nieanonimowe sieci wymiany plików zaprezentujemy na przykładzie dwóch najpopularniejszych: *eDonkey2000* oraz *BitTorrent*.

eDonkey2000

eDonkey2000 [25] jest siecią scentralizowaną, wykorzystującą serwery, które przechowują informacje o wszystkich użytkownikach oraz o plikach przez nich udostępnianych. Każdy plik w sieci jest jednoznacznie identyfikowany przez URI zawierający jego rozmiar oraz sumę kontrolną MD4 [12]. Protokół działania jest dość prosty: klient, który chce pobrać plik, łączy się z serwerem, wysyła URI pliku, a następnie dostaje listę adresów IP innych użytkowników, którzy ten plik posiadają. Następnie klient łączy się bezpośrednio z każdym z tych użytkowników i prosi o przesłanie mu odpowiednich części pliku. Posiadacz pliku utrzymuje kolejkę

klientów proszących o jego pliki i realizuje żądania w zdefiniowanej kolejności (w najprostszym przypadku jest to kolejność ich nadchodzenia).

Oryginalną aplikacją korzystającą z sieci *eDonkey2000* był klient o tej samej nazwie stworzony przez firmę *Metamachine*, jednak jego rozwój został wstrzymany. Aktualnie najpopularniejszym klientem tej sieci jest program *eMule* [28], rozprowadzany na licencji GNU [26].

Głównymi zaletami sieci *eDonkey2000* są: duża w porównaniu z anonimowymi sieciami szybkość pobierania plików oraz duża baza zidentyfikowanych adresów URI plików, dostępnych na specjalnie w tym celu tworzonych serwisach WWW.

Sieć *eDonkey2000* nie zapewnia swoim użytkownikom żadnej anonimowości i nie istnieje implementacja jej klienta korzystająca z sieci anonimowej.

BitTorrent

BitTorrent [2], dzieło programisty Brama Cohena, którego pierwsza wersja klienta była prostą aplikacją napisaną w mało popularnym wówczas języku programowania *Python* [33], urosło do aktualnie najpopularniejszej i najchętniej stosowanej w wielu zastosowaniach sieci wymiany plików. Geniusz *BitTorrenta* polegał na tym, że zerwał on z koncepcją jednego centralnego serwera wszystkich plików. Zamiast tego w *BitTorrencie* małe sieci tworzone są wokół pojedynczego pliku, a nad wszystkim pieczę sprawują dedykowane serwery kontrolne (ang. *trackers*). Dzięki podzieleniu sieci na bardzo małe, niezależne fragmenty, praktycznie wyeliminowano znany z *eDonkey2000* problem braku skalowalności, co drastycznie zwiększyło szybkość pobierania plików przez klientów tej sieci. Z drugiej strony, sieć *BitTorrent* nie zapewnia sama żadnych algorytmów wyszukujących – rolę tę przejęły niezależne serwisy WWW.

Obecnie najpopularniejszymi klientami tej sieci są: napisany w Javie *Azureus* [23] oraz niewielki, bo napisany w języku C++, *µTorrent* [38]. Godny odnotowania jest fakt, iż *Azureus* wspiera przesyłanie plików w sieci anonimowej *I2P* oraz komunikację z serwerami kontrolnymi przez sieć anonimową *Tor*. Przesyłanie plików, ze względu na duże obciążanie dedykowanych serwerów pośredniczących, zostało w sieci *Tor* zabronione.

2.3. Założenia projektu

Analiza zapotrzebowań (patrz rozdział 2.1) oraz istniejących już rozwiązań doprowadziła do następujących wniosków:

- istniejące sieci punkt-do-punktu służące do wymiany treści nie są zaprojektowane z myślą o zapewnianiu anonimowości;
- istniejące sieci zapewniające anonimowość umożliwiają wymianę treści, ale prędkość tej wymiany jest niezadowalająca, gdyż zastosowane rozwiązania istotnie spowalniają pobieranie plików; ponadto, istniejące sieci anonimowe zapewniają anonimowość pełną – dla nadawcy i odbiorcy treści, nas jednak interesuje wyłącznie anonimowość nadawcy;
- brakuje „złotego środka” – aplikacji, która z jednej strony zapewniałaby wystarczającą anonimowość dla nadawców treści, a z drugiej nie powodowała nadmiernego narzutu na wydajność.

Powyższe wnioski doprowadziły do powstania założeń projektu, którego realizacja jest częścią niniejszej pracy.

Projekt zakładał zaprojektowanie i napisanie aplikacji, która:

- umożliwia przesyłanie treści pomiędzy użytkownikami Internetu;
- zachowuje anonimowość nadawcy treści przy jak najmniejszych kosztach czasu procesorów i łącz internetowych użytkowników;
- posiada wygodny graficzny interfejs użytkownika.

Realizacja projektu spełniającego te założenia opisana została w rozdziale 3.

Rozdział 3

Realizacja projektu

3.1. Możliwości zapewniania anonimowości

W trakcie rozważań nad realizacją projektu okazało się, że anonimowość dla udostępniających treści można zapewnić na dwa sposoby: przez fałszowanie adresu IP nadawcy w nagłówkach pakietu *UDP* oraz przez tunelowanie. Poniżej zostaną przedstawione oba pomysły.

Fałszowanie adresu IP nadawcy

Kiedy komputer otrzymuje pakiet z Internetu, jedynym sposobem, by poznać, od kogo ten pakiet pochodzi, jest sprawdzenie pola „adres źródłowy" w nagłówku *IP* tego pakietu (patrz rozdział 1.1). Adres ten jest potrzebny komputerowi, aby wiedział, komu ma odpowiedzieć. Jeśli więc będziemy potrafili w inny sposób poinformować odbiorcę, w jaki sposób odpowiedzieć na otrzymaną informację, to możemy adres źródłowy nadawcy podmienić na losową liczbę lub nawet wyzerować. Ze względu na to, że w protokole TCP pole nadawcy jest konieczne do jego prawidłowego działania (konkretnie do automatycznego potwierdzania otrzymania pakietów), jedynym protokołem, który można zastosować, jest protokół UDP.

Możliwa realizacja takiego pomysłu wygląda następująco:

- Wszystkie węzły są podłączone do pewnej istniejącej sieci anonimowej, takiej jak np. *Tor*.
- Wyszukiwanie pliku odbywa się w standardowy dla sieci anonimowych sposób – poprzez pośredników.
- W momencie, w którym właściciel W pliku P otrzyma od swojego sąsiada informację, że węzeł A właśnie tego pliku P poszukuje, odpowiada mu przez wszystkich pośredników, przez których przyszło zapytanie.
- Węzeł A, gdy otrzyma informację, że węzeł W posiada plik P, zaczyna wysyłać do niego, nadal przez pośredników, prośby o przesłanie konkretnych części pliku.
- Właściciel W, gdy otrzyma prośbę o konkretną część pliku P od węzła A, przesyła węzłowi A tę część bezpośrednio, używając protokołu UDP z wyzerowanym adresem źródłowym. Komunikacja jest bardzo szybka, a węzeł A nie zna adresu IP węzła W.
- Węzeł A otrzymuje (lub nie – gdy są błędy w komunikacji) pakiet od właściciela W i wysyła do niego (przez sąsiadów – gdyż jest to jedyna droga, jaką potrafi się komunikować z W) informację, czy fragment pliku doszedł w nienaruszonym stanie. Jeśli nie, to W może ponowić próbę wysłania pliku.

Zauważmy, że rozwiązanie to ma wszystkie cechy, których wymagamy – nadawca pozostaje anonimowy, a mimo to komunikacja w jedną stronę z odbiorcą odbywa się w najszybszy możliwy sposób – przez bezpośrednie połączenie. Ponieważ właśnie w stronę odbiorcy będą przechodziły największe pakiety, uzyskujemy ogromny zysk wydajności w porównaniu z innymi sieciami anonimowymi.

Realizacja sieci anonimowej za pomocą fałszowania adresu źródłowego UDP wydaje się świetnym pomysłem, ale napotyka poważne problemy, o czym za chwilę.

Tunelowanie

Drugim sposobem zapewnienia anonimowości nadawcy jest tworzenie tuneli. Ponieważ chcemy zapewniać jedynie anonimowość nadawcy, tylko dla niego będą takie tunele tworzone – tunele dla odbiorców są zbędne i nie trzeba się nimi zajmować.

Wyszukiwanie w sieci z tunelami odbywa się w standardowy sposób – przez sieć pośredników. Kiedy nadawca otrzyma prośbę o przesłanie pliku, wysyła żądaną część przez jeden ze swoich tuneli, a ostatni pośrednik w tunelu wysyła wiadomość bezpośrednio do odbiorcy. Rozwiązanie to zostanie opisane dokładniej w późniejszych rozdziałach.

Testy i wnioski

Pierwotnym założeniem projektującego było stworzenie aplikacji, która wykorzystuje fałszowanie adresu źródłowego UDP, gdyż wydawało się to idealnym rozwiązaniem. Jednakże okazało się, że rutery większości dostawców Internetu na świecie, w tym wszystkich dostawców w Polsce, zostały zaprogramowane, by nie pozwalać na wysyłanie pakietów o wyzerowanym lub różnym od faktycznego adresie nadawcy. Powodem zablokowania tej możliwości było używanie dokładnie tego samego sposobu do przeprowadzania ataków tzw. *rozproszonej odmowy dostępu (ang. DDoS, Distributed Denial of Service)*, czyli zawieszania serwerów sieciowych przez wysyłanie do nich jednocześnie bardzo wielu zapytań z różnych komputerów, co powodowało przeciążenie serwera i niemożliwość obsługiwania użytkowników. Aby zwiększyć skuteczność takiego ataku, atakujący podmieniali swoje adresy IP w pakietach, co dawało im tę przewagę, że nie musieli reagować na odpowiedź serwera – serwer, przekonany że wiadomość przyszła skądinąd, wysyłał odpowiedź na podany adres i nigdy nie doczekiwał się reakcji.

Serwis *ANA Spoofer Project* [22] podjął się zmierzenia, ile komputerów na świecie jest jeszcze w stanie wysyłać pakiety z podmienionym adresem nadawcy. Eksperyment polegał na tym, że poproszono użytkowników, aby pobrali i uruchomili aplikację, która próbowała wysłać pakiety z podmienionym adresem źródłowym na serwer projektu. Wyniki są niekorzystne: jeszcze tylko $16,2\%$ sieci komputerowych zezwala na wysyłanie takich pakietów, a liczba ta wciąż maleje. W samej Polsce tylko jeden mały dostawca Internetu (nie podano jaki) zezwala na ich przesyłanie. Sama aplikacja testowa, aby zadziałać w środowisku *Microsoft Windows XP*, musi instalować specjalny sterownik, gdyż sterownik sieciowy w tym systemie operacyjnym również blokuje wysyłanie fałszywych pakietów.

Wobec napotkanych trudności i pesymistycznych wyników testów, zdecydowano się na realizację anonimowości przez tunelowanie. Szczegóły realizacji opisane są w kolejnych rozdziałach.

3.2. Przygotowanie

Wstępem do realizacji projektu był wybór technologii i narzędzi pomocniczych oraz projekt protokołów i algorytmów.

3.2.1. Język programowania

Jako język programowania wybrany został *Python* [33]. *Python* należy do najnowszej generacji obiektowych języków programowania, zapewnia wsparcie dla programisty w postaci automatycznego zarządzania pamięcią, obsługi wyjątków i wygodnego programowania wielowątkowego. Jest dostępny dla wielu systemów operacyjnych. Jedną z największych zalet *Pythona* jest ogromna baza bibliotek dołączanych do samej dystrybucji oraz tworzonych przez niezależnych programistów. Charakterystyczną cechą składni *Pythona* jest brak statycznej kontroli typów oraz oznaczanie bloków programu za pomocą wcięć – powoduje to, iż jego kod należy do najbardziej czytelnych ze wszystkich obecnie dostępnych języków.

Przy rozważaniu środowiska programistycznego, wybór padł na *Eclipse* [24] wraz z wtyczką *PyDev* [30] oraz programem do kontroli jakości kodu *PyLint* [31]. Za wyborem przemawiały przede wszystkim: dojrzałość *Eclipse'a* oraz fakt, iż wszystkie wymienione narzędzia rozpowszechniane są na licencjach Open Source.

3.2.2. Biblioteki

Wbudowane biblioteki *Pythona* dostarczyły prawie wszystkich narzędzi potrzebnych do zbudowania aplikacji, jednak trzeba było skorzystać z kilku zewnętrznych bibliotek.

Biblioteka *wxPython* [37], będąca wersją biblioteki *wxWidgets* dla Pythona, służy do tworzenia aplikacji z graficznym interfejsem użytkownika, które wyglądają atrakcyjnie pod wieloma systemami operacyjnymi, m.in. pod *Windows*, *Linuksem* i *Mac OS X*.

Moduł *Python Cryptography Toolkit* [32] zapewnił potrzebne wsparcie dla wszystkich funkcji kryptograficznych, takich jak generowanie liczb pierwszych, szyfrowanie symetryczne i funkcje skrótu.

Biblioteka *Python Remote Objects* [34] posłużyła za warstwę komunikacji sieciowej pomiędzy aplikacjami. Zapewnia ona przezroczysty interfejs synchronicznego, zdalnego wywoływania procedur obiektów znajdujących się na innych komputerach w sieci. Jej zastosowanie oszczędza programiście konieczności oprogramowania gniazd sieciowych i używania niskopoziomowego kodu systemowego.

3.2.3. Architektura sieci punkt-do-punktu

Po dogłębnej analizie istniejących sieci anonimowych okazało się, iż żadna z nich nie spełnia do końca zapotrzebowań tworzonej aplikacji. Najbliższa ideałowi sieć *I2P* nie wspierała połączenia z żadną biblioteką zdalnego wywoływania procedur, a poza tym była zbyt rozbudowana jak na potrzeby projektu. Zaprojektowana więc została własna sieć punkt-do-punktu, która pozwala na połączenia między sąsiadami, tworzenie *tuneli* dla użytkowników publikujących treści, obsługę wyszukiwania oraz wsparcie dla biblioteki zdalnego wywoływania procedur. Dokładne działanie sieci przedstawione jest w rozdziale 3.3.3.

3.2.4. Identyfikacja treści w sieci

Podstawową jednostką treści w zaprojektowanej sieci jest plik. Plik jest jednoznacznie identyfikowany przez parę: rozmiar pliku (w bajtach) oraz jego sumę kontrolną, uzyskaną za pomocą algorytmu MD4 [12]. Wybór algorytmu spowodowany był faktem, iż najpopularniejsza nieanonimowa sieć punkt-do-punktu *eDonkey2000* również korzysta z MD4 jako algorytmu liczenia sum kontrolnych. Umożliwia to ewentualne powiązanie ze sobą obu sieci w przyszłości.

3.3. Aplikacja „Sandstorm”

Wynikiem prac nad projektem jest aplikacja o nazwie *Sandstorm* (od burzy piaskowej, w której nic nie widać – odniesienie do zapewnianej anonimowości).

3.3.1. Opis funkcjonalności

Sandstorm pozwala użytkownikowi na:

- podłączenie się do sieci punkt-do-punktu – jest do tego potrzebna znajomość co najmniej jednego użytkownika, który już jest podłączony; sieć, do której podłącza się użytkownik, została stworzona specjalnie dla aplikacji *Sandstorm* i nie jest połączona w żaden sposób z innymi sieciami punkt-do-punktu;
- wyszukanie pliku w sieci – odbywa się przez podanie przez użytkownika *URI* pliku, który jednoznacznie identyfikuje jego treść; *URI* ma postać:
  `ss://<nazwa_pliku>|<rozmiar_w_bajtach>|<suma_kontrolna_MD4>|/` ;
- pobranie pliku na dysk – użytkownik może wybrać, do jakiego katalogu zostanie ściągnięty plik; pobieranie plików nie zapewnia anonimowości pobierającemu;
- udostępnianie plików – użytkownik wybiera katalogi z plikami, które chce udostępnić innym użytkownikom; zapewniana jest anonimowość udostępniającego;
- konfigurację połączenia – użytkownik może ustalić:
  - liczbę sąsiadów (bezpośrednich połączeń z innymi użytkownikami);
  - liczbę tuneli (służących do anonimowego udostępniania plików);
  - minimalną i maksymalną długość tuneli.

3.3.2. Przykład działania

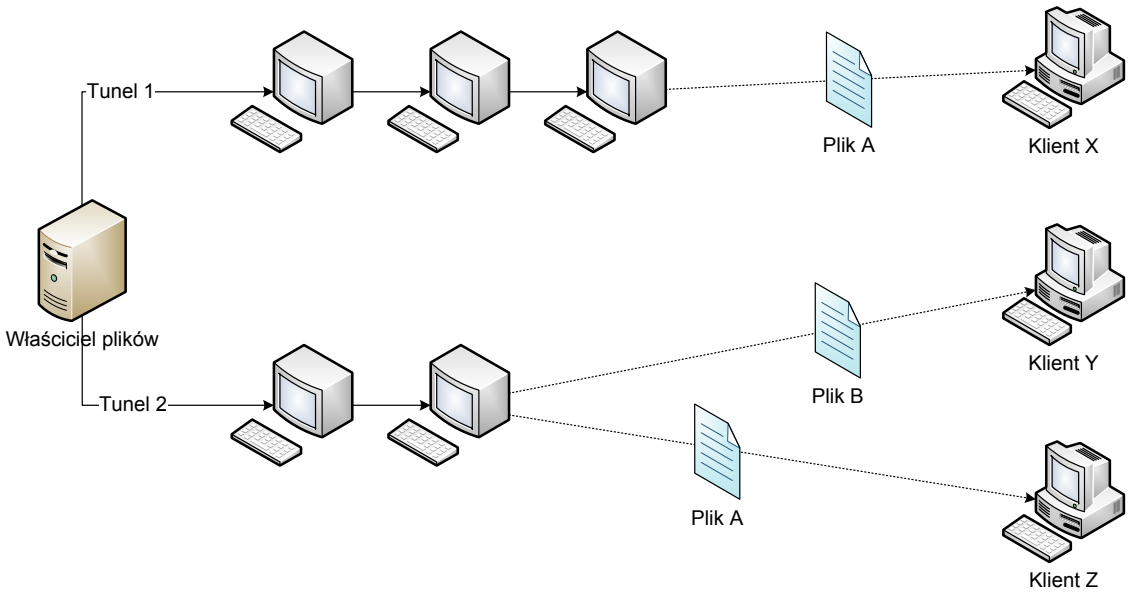
Działanie sieci *Sandstorm* pokażemy na przykładzie. Rysunek 3.1 przedstawia sytuację, w której trzech klientów: X, Y i Z pobiera pliki od ich właściciela (serwer po lewej stronie)[1]. Właściciel, na potrzeby udostępniania plików, stworzył dwa tunele: *Tunel 1* o długości 3 oraz *Tunel 2* o długości 2. Klienci X i Z pobierają plik A, natomiast klient Y pobiera plik B. Dla każdego klienta, właściciel wybrał tunel, przez który będzie się z nim komunikował. Poszczególne fragmenty pliku przechodzą przez wszystkie węzły tunelu zanim zostaną wysłane bezpośrednio od ostatniego pośrednika w tunelu do klienta.

Zauważmy, że żaden z klientów nie jest bezpośrednio połączony z właścicielem pliku, a więc nie zna jego tożsamości. Właściciel plików zna natomiast tożsamość klientów – musi ich adres IP przekazać ostatniemu węzłowi w tunelu, by ten wiedział, dokąd wysłać plik.

3.3.3. Zasada działania

W tym rozdziale przedstawione zostaną algorytmy, których *Sandstorm* używa do realizacji swojej funkcjonalności.

[1] Rysunki przedstawiające węzły uczestniczące w tej sytuacji są różne ze względu na różne role, które w tej sytuacji spełniają, każdy z nich jest jednak taką samą aplikacją kliencką, o tych samych uprawnieniach.



Rysunek 3.1: Przykładowa sytuacja w sieci *Sandstorm*.

Nawiązywanie połączenia między węzłami

Sandstorm korzysta z własnej, niezależnej od innych, sieci punkt-do-punktu. Podstawą sieci punkt-do-punktu są połączenia między węzłami (punktami). Procedura nawiązywania połączenia między węzłem A a węzłem B wygląda następująco:

- A: Nawiąż połączenie TCP do węzła B z wykorzystaniem funkcji biblioteki *Python Remote Objects*;
- B: Serwer *Python Remote Objects*, pracujący na osobnym wątku, przyjmuje połączenie od A i tworzy kolejny wątek specjalnie dla tego połączenia.
- A: Jeżeli komunikujesz się z węzłem B po raz pierwszy, to zastosuj algorytm Diffiego-Hellmana (patrz rozdział 1.4), którego wynikiem będzie 256-bitowy klucz do szyfrowania symetrycznego $k_{A,B}$;
- Od tej pory każdy komunikat przesyłany między A i B będzie szyfrowany algorytmem AES [3] z kluczem $k_{A,B}$.

Podłączenie do sieci

Scenariusz: węzeł A chce połączyć się z siecią, do której jest już podłączony węzeł B.

1. A: Pobierz od użytkownika adres IP lub nazwę komputera oraz port, którego używa już podłączony do sieci węzeł B.
2. A: Spróbuj nawiązać połączenie z podanym komputerem. Jeśli się nie udało, wróć do punktu 1.
3. A: Poproś nowo podłączony komputer, aby został twoim sąsiadem.

4. B: Sprawdź, czy nie masz już zbyt wielu sąsiadów. Jeśli tak, to odrzuć prośbę. Jeśli nie, to zaakceptuj prośbę i dodaj sąsiada A do listy swoich sąsiadów.

5. A: Jeśli B zaakceptował prośbę, dodaj go do listy swoich sąsiadów. W przeciwnym wypadku wróć do punktu 1.

Stworzenie tunelu

Scenariusz: użytkownik A chce stworzyć kolejny tunel.

1. A: Wylosuj liczbę $N \in < T_{min}, T_{max} >$, gdzie T_{min} i T_{max} oznaczają opcje w konfiguracji: odpowiednio minimalną i maksymalną długość tunelu;

2. A: Wygeneruj *unikatowy identyfikator* tunelu ID_T.

3. A: Oznacz wszystkich sąsiadów jako niewypróbowanych.

4. A: Wylosuj jednego niewypróbowanego sąsiada S. Jeśli taki nie istnieje, to zakończ algorytm z błędem.

5. A: Oznacz sąsiada S jako wypróbowanego.

6. A: Poproś sąsiada o wygenerowanie tunelu o długości $N - 1$, o identyfikatorze ID_T. Jeśli sąsiad przekazał błąd, to wróć do punktu 4.

7. A: Utwórz tunel o długości N przez dodanie siebie jako ostatniego ogniwa tunelu przekazanego przez sąsiada.

8. A: Przekaż stworzony tunel jako wynik algorytmu. Dodaj stworzony tunel do listy tuneli.

Spójrzmy na algorytm działania sąsiada S, który otrzymuje prośbę o stworzenie tunelu:

1. S: Odbierz prośbę o stworzenie tunelu o długości N, o identyfikatorze ID_T.

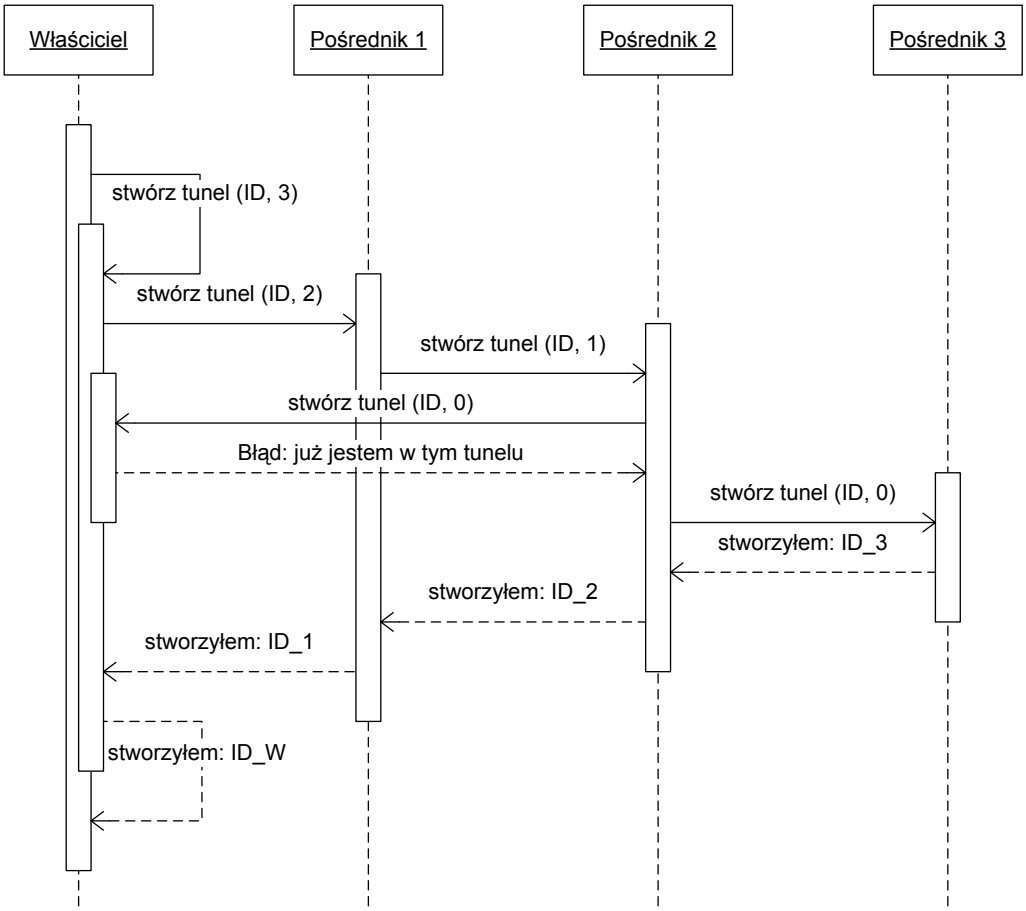
2. S: Sprawdź, czy nie obsługujesz już tunelu ID_T. Jeśli tak, to przekaż błąd.

3. S: Jeśli $N == 0$, to zaakceptuj prośbę i jako rezultat przekaż tunel, którego jesteś jedynym elementem. Dodaj przekazany tunel do listy obsługiwanych tuneli.

4. S: W przeciwnym wypadku, zastosuj kroki od 3 do 8 algorytmu dla użytkownika A.

Rysunek 3.2 przedstawia diagram przepływu UML [9] obrazujący przykładową procedurę tworzenia tunelu.

Właściciel tunelu, aby zbudować tunel o długości 3, wybiera pierwszego pośrednika (*Pośrednik 1*) wśród swoich sąsiadów i wysyła mu prośbę o stworzenie tunelu o długości 2, razem z globalnym identyfikatorem tunelu. *Pośrednik 1* wybiera swojego sąsiada (*Pośrednik 2*) i wysyła mu prośbę o stworzenie tunelu o długości 1. *Pośrednik 2* losuje swojego sąsiada – okazuje się, że wylosował właściciela tunelu. Wysyła mu prośbę o bycie kolejnym pośrednikiem i dostaje w rezultacie błąd – ten sam węzeł nie może być dwukrotnie pośrednikiem w tym samym tunelu. W zaistniałej sytuacji *Pośrednik 2* wybiera innego sąsiada, *Pośrednika 3*, i prosi go o stworzenie tunelu o długości 0. *Pośrednik 3* akceptuje prośbę i zwraca lokalny identyfikator tunelu ID_3. Każdy kolejny pośrednik tworzy swój własny identyfikator tunelu i zwraca go poprzednikowi. W końcu właściciel tunelu otrzymuje od *Pośrednika 1* jego identyfikator ID_1 i generuje własny identyfikator ID_W. Procedura tworzenia tunelu kończy się sukcesem.



Rysunek 3.2: Przykład tworzenia tunelu w programie *Sandstorm*.

Wyszukiwanie pliku

Scenariusz: użytkownik A chce wyszukać i pobrać plik P o identyfikatorze $Uri(P)$.

1. A: Sprawdź, czy masz plik o identyfikatorze $Uri(P)$ wśród udostępnianych plików. Jeśli tak, to zakończ algorytm z błędem.
2. A: Wygeneruj *unikatowy identyfikator* zapytania ID_P.
3. A: Dla każdego sąsiada S, wyślij do S prośbę o plik o identyfikatorze $Uri(P)$ dla węzła A wraz z identyfikatorem zapytania ID_P.
4. A: Uruchom osobny wątek odpowiedzialny za pobieranie pliku.

Algorytm sąsiada S otrzymującego zapytanie wygląda następująco:

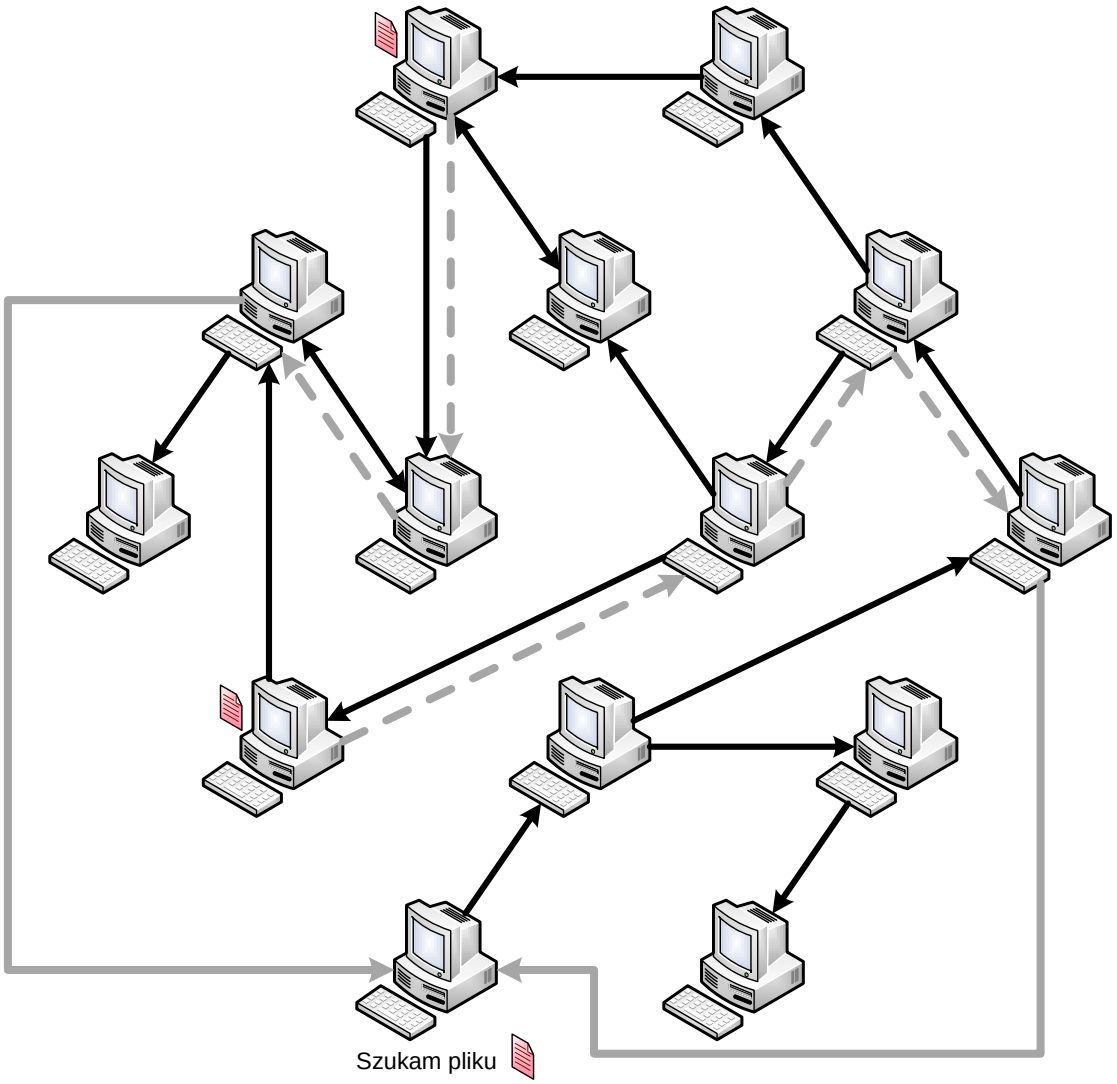
1. S: Odbierz od sąsiada S' prośbę o plik o identyfikatorze $Uri(P)$ dla węzła A wraz z identyfikatorem zapytania ID_P.
2. S: Sprawdź, czy już nie obsługiwałeś zapytania ID_P. Jeśli tak, to zakończ algorytm.
3. S: Dodaj identyfikator zapytania ID_P do listy obsłużonych zapytań.

4. S: Dla każdego sąsiada S'' oprócz S', wyślij do S'' prośbę o plik o identyfikatorze $Uri(P)$ dla węzła A wraz z identyfikatorem zapytania ID_P.

5. S: Sprawdź, czy masz plik o identyfikatorze $Uri(P)$ wśród udostępnianych plików. Jeśli nie, to zakończ algorytm.

6. S: Przez jeden ze swoich tuneli wyślij odpowiedź do węzła A zawierającą identyfikator pliku $Uri(P)$ oraz identyfikator zapytania ID_P. Zauważmy, że nie jest wysyłany adres węzła S.

Rysunek 3.3 przedstawia przykładowy proces wyszukiwania pliku w sieci *Sandstorm*.



Rysunek 3.3: Przykładowe wyszukiwanie pliku w sieci *Sandstorm*. Posiadacze pliku oznaczeni są ikoną pliku. Czarne strzałki oznaczają połączenia między sąsiadami. Szare przerywane strzałki to odpowiedzi przesyłane przez tunel. Szare ciągłe strzałki to odpowiedzi przesłane bezpośrednio od ostatniego pośrednika w tunelu do odbiorcy.

W przedstawionej sytuacji węzeł na dole rozpoczyna poszukiwanie pliku. Dane pliku i adres IP poszukującego są przekazywane przez kolejnych sąsiadów, aż przejdą przez całą

sieć. Posiadacze pliku (oznaczeni ikoną pliku) komunikują się z poszukującym przez tunele rozpoczynające się od ich sąsiadów (szare przerywane strzałki). Ostatni pośrednik w tunelu przesyła wiadomość bezpośrednio do poszukującego (szare ciągłe strzałki).

Tunelowanie wiadomości od właściciela

Zasada działania tuneli została przedstawiona w rozdziale 1.4. Tu opiszemy następujący scenariusz: węzeł A chce przesłać do węzła B wiadomość W przez tunel.

1. A: Wylosuj tunel T z listy własnych tuneli. Jeśli lista jest pusta, to zakończ algorytm z błędem.
2. A: Do pierwszego pośrednika P_1 z tunelu T wyślij prośbę o przekazanie węzłowi B wiadomości W.
3. A: Odbierz odpowiedź bądź informację o błędzie od pośrednika P_1.
4. A: Jeśli nastąpiła awaria tunelu, to usuń go z listy swoich tuneli i wróć do punktu 1.

Algorytm dowolnego z pośredników P_i w tunelu wygląda następująco:

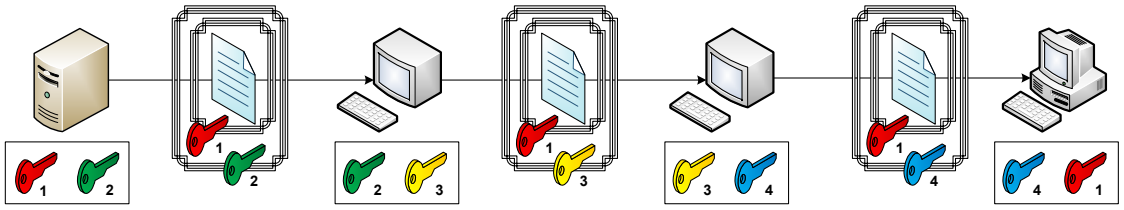
1. P_i: Odbierz od poprzednika w tunelu P_{i-1} prośbę o przekazanie węzłowi B wiadomości W.
2. P_i: Jeśli jesteś ostatnim pośrednikiem w tunelu:
   (a) Nawiąż bezpośrednie połączenie sieciowe z węzłem B. Jeśli nie powiodło się, to przekaż poprzednikowi P_{i-1} błąd połączenia.
   (b) Wyślij wiadomość W do węzła B i poczekaj (synchronicznie) na odpowiedź.[2]
   (c) Przekaż odpowiedź poprzednikowi P_{i-1}.

   Jeśli natomiast nie jesteś ostatnim pośrednikiem w tunelu:

   (a) Wyślij do pośrednika P_{i+1} prośbę o przekazanie węzłowi B wiadomości W. Jeśli połączenie z pośrednikiem nie powiedzie się, to uznaj tunel za uszkodzony, usuń go z listy swoich tuneli i przekaż poprzednikowi P_{i-1} błąd awarii tunelu.
   (b) Odbierz odpowiedź od pośrednika P_{i+1}. Jeśli jest to informacja o awarii tunelu, to usuń tunel z listy swoich tuneli. Przekaż odpowiedź poprzednikowi P_{i-1}.

Zauważmy, że z punktu widzenia pośrednika P_1 właściciel tunelu A jest po prostu kolejnym poprzednikiem – P_1 nie wie więc, kto jest właścicielem tunelu.

Przy wysyłaniu wiadomości przez tunel do odbiorcy specjalna pośrednia warstwa komunikacji sprawdza, czy z tym odbiorcą komunikujemy się po raz pierwszy. Jeśli tak jest, to pierwszymi dwoma wymienionymi komunikatami są komunikaty algorytmu Diffiego-Hellmana, których wynikiem jest klucz szyfrowania symetrycznego między właścicielem tunelu a odbiorcą komunikatu wysyłanego przez tunel. Od tej pory wszystkie wiadomości wysyłane przez właściciela tunelu do tego odbiorcy szyfrowane są ich kluczem symetrycznym. Takie dodatkowe szyfrowanie (poza szyfrowaniem między każdymi dwoma pośrednikami w tunelu) ma na celu ukrycie treści wiadomości przed pośrednikami w tunelu.



Rysunek 3.4: Przykład szyfrowania w tunelu. Ramki naokoło plików oznaczają zaszyfrowaną zawartość. Pod każdym węzłem pokazana jest lista kluczy szyfrujących, jakie on posiada.

W przedstawionej na rysunku 3.4 przykładowej sytuacji, serwer (po lewej) przesyła fragment pliku do klienta (po prawej) przez tunel o długości 2.

Plik zostaje najpierw zaszyfrowany kluczem ustalonym przez serwer i klienta (klucz 1). Następnie wynik tego szyfrowania jest szyfrowany kluczem serwer-pierwszy pośrednik (klucz 2). Pierwszy pośrednik odbiera pakiet, rozszyfrowuje go kluczem 2, a następnie szyfruje kluczem pierwszy-drugi pośrednik (klucz 3) i przesyła dalej. Drugi pośrednik rozszyfrowuje pakiet kluczem 3, pakuje go kluczem drugi pośrednik-klient (klucz 4) i wysyła do klienta. Klient rozszyfrowuje plik najpierw kluczem 4, a następnie kluczem 1.

Tunelowanie wiadomości do właściciela

Tunelowanie wiadomości do właściciela tunelu wygląda podobnie do tunelowania od właściciela, z tym że wysyłający użytkownik nie musi określać odbiorcy wiadomości – jest nim właściciel tunelu i jest on jednoznacznie wyznaczany przez węzeł będący końcem tunelu oraz identyfikator tunelu.

Pobieranie pliku

Osobny wątek aplikacji użytkownika A oczekuje na odpowiedzi na zapytanie o plik, a osobny zarządza jego pobieraniem. Do utrzymywania informacji o tym, skąd można pobrać ściągany plik, tworzona jest tzw. *lista źródeł* dla każdego pliku. Zawiera ona pary (W, T), gdzie T jest identyfikatorem tunelu prowadzącego do właściciela pliku, a W jest ostatnim węzłem tego tunelu.

Spójrzmy najpierw, co się dzieje, gdy nadchodzi odpowiedź na zapytanie o plik:

1. A: Odbierz od węzła W, będącego końcem pewnego tunelu T, odpowiedź zawierającą identyfikator pliku $Uri(P)$ oraz identyfikator zapytania ID_P.

2. A: Sprawdź, czy oczekiwałeś odpowiedzi na takie zapytanie. Jeśli nie, to zakończ algorytm.

3. A: Dodaj parę (W, T) do listy źródeł pliku P.

Wątek pobierający plik P wykonuje następujące czynności:

1. A: Wybierz część pliku p_i, którą będziesz chciał pobrać. Jeśli wszystkie części są już pobrane, to zakończ wątek i poinformuj użytkownika o sukcesie.

[2]Pośrednik może czekać synchronicznie, ponieważ polecenie przesłania przez tunel jest realizowane w osobnym wątku aplikacji.

2. *A*: Wylosuj parę (W, T) z listy źródeł pliku P. Jeśli lista źródeł jest pusta, to uśpij wątek na kilka sekund i spróbuj ponownie. Jeśli lista źródeł jest pusta przez dłuższy czas, uruchom ponownie wątek wyszukujący plik w sieci w celu znalezienia nowych właścicieli pliku.

3. *A*: Wyślij do węzła W prośbę, aby przez tunel T przekazał jego właścicielowi prośbę o część p_i pliku o identyfikatorze $Uri(P)$.

4. *A*: Odbierz odpowiedź od węzła W. Jeśli odpowiedź jest negatywna lub nastąpił błąd komunikacji, to usuń źródło (W, T) z listy źródeł i wróć do punktu 2.

5. *A*: Zapisz odebrany fragment pliku na dysk, oznacz fragment p_i jako pobrany i wróć do punktu 1.

Pod koniec procesu pobierania pliku sprawdzana jest jego suma kontrolna MD4. Porównywana jest ona z sumą zawartą w URI tego pliku. Pozwala to wykryć błędy w transmisji. Jeśli plik został pobrany z błędem, to można spróbować pobrać go ponownie.

3.4. Architektura

Ten rozdział poświęcony jest opisowi architektury aplikacji *Sandstorm*.

3.4.1. Podział na pakiety

Aplikacja *Sandstorm* podzielona jest na trzy główne pakiety: **hive**, **ui** oraz **common**. W języku *Python* pakiety są reprezentowane przez katalogi zawierające specjalny plik inicjujący `__init__.py` oraz inne pliki języka *Python*, z których każdy jest osobnym modułem pakietu.

Pakiet hive

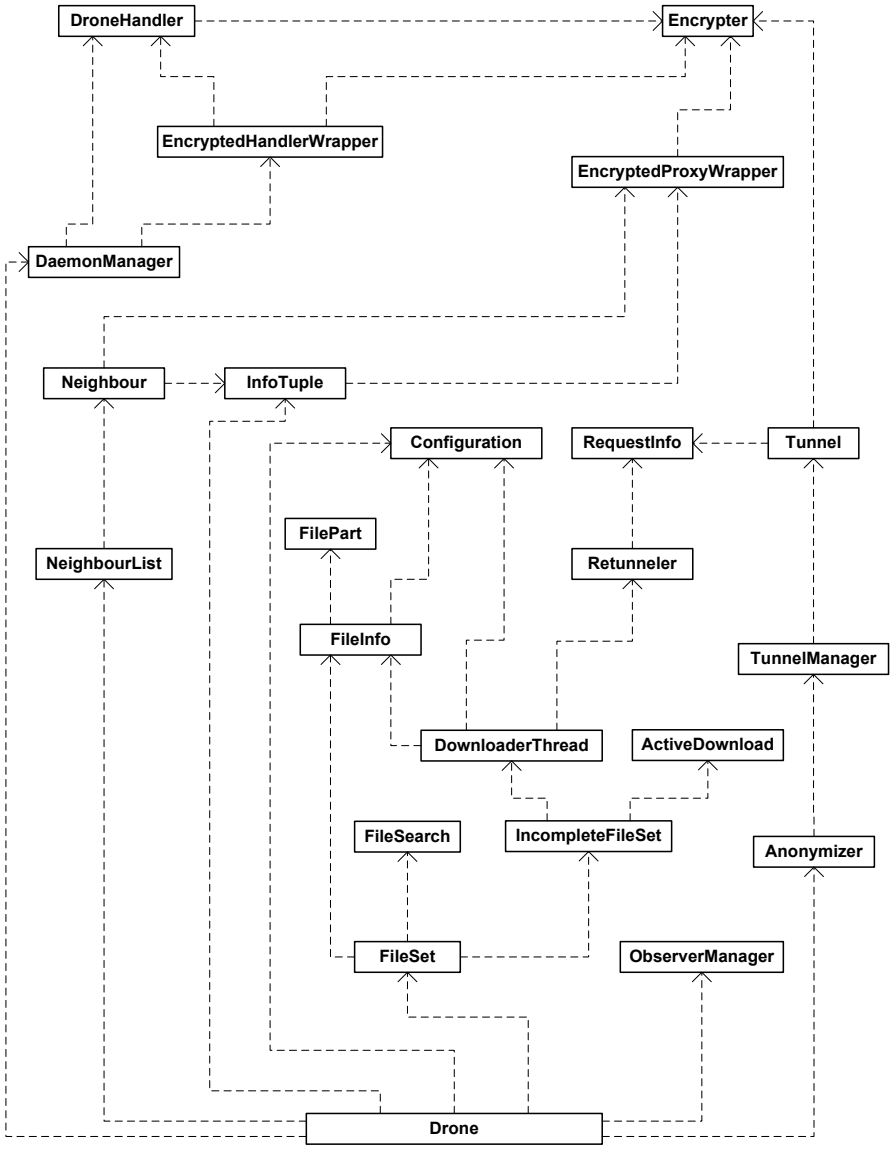
Pakiet **hive** jest najważniejszym pakietem aplikacji *Sandstorm*. W nim mieszczą się moduły silnika programu.

Najważniejszą klasą z pakietu **hive** jest **Drone**. Każdy obiekt klasy Drone jest ekwiwalentem jednego węzła w sieci *Sandstorm*. Drone korzysta z pozostałych klas pakietu **hive** do realizacji swojej funkcjonalności węzła. To z obiektem klasy Drone komunikuje się pakiet **ui**, aby pokazać informacje użytkownikowi. Główne moduły, z których korzysta klasa Drone, to:

- **Configuration** – dynamiczna konfiguracja węzła. Ustala się w niej takie parametry pracy węzła, jak długość tuneli, udostępniane katalogi itp. Konfiguracja oparta jest na uniwersalnym systemie zapamiętywania **wxConfig**, który, w zależności od systemu operacyjnego, wybiera najlepsze miejsce do przechowywania informacji – w przypadku systemu *Microsoft Windows XP* jest to rejestr systemowy.

- **NeighbourList** – obiekty tej klasy odpowiedzialne są za utrzymywanie listy sąsiadów węzła. Lista jest przystosowana do jednoczesnego używania przez wiele wątków i posiada odpowiednie mechanizmy synchronizacji. Potrafi również zweryfikować żywotność przechowywanych sąsiadów oraz uzupełnić ich liczbę, gdy jest ich zbyt mało.

- **ObserverManager** – implementuje Pythonową wersję wzorca projektowego *obserwator* [10] w celu zapewnienia komunikacji z interfejsem użytkownika. Wzorzec obserwator



Rysunek 3.5: Diagram zależności między modułami pakietu **hive** w programie *Sandstorm*.

polega na tym, że w momencie, gdy obserwowany obiekt (Drone) zmienia swój stan, ObserverManager wysyła informację o tej zmianie do wszystkich obserwatorów. Każdy z obserwatorów może wtedy zareagować odpowiednio do sytuacji i samodzielnie pobrać potrzebne dane od obserwowanego obiektu.

- **FileSet** – przechowuje listę pobieranych i udostępnianych plików oraz zarządza nią. Implementuje takie funkcje, jak rozpoczęcie wyszukiwania pliku czy dodanie katalogu do listy udostępnianych katalogów.

- **DaemonManager** – zarządza wątkiem demona *Python Remote Objects*, służącego do odbierania komunikatów od innych węzłów i przetwarzania ich.

- **Anonymizer** oraz **TunnelManager** – służą do zapewniania anonimowości. Ich zadaniem jest utrzymywanie odpowiedniej liczby tuneli, upewnianie się, że są one aktywne,

a także tunelowanie wiadomości i przekazywanie jej dalej w tunelu, jeśli węzeł jest czyimś pośrednikiem.

Rysunek 3.5 przedstawia diagram UML zależności między modułami w pakiecie **hive**.

Pakiet ui

Pakiet **ui** zawiera klasy tworzące interfejs użytkownika. Zrealizowany on został według wzorca projektowego Model-Widok-Koordynator [8], polegającego na rozdzieleniu silnika aplikacji (modelu) od jej interfejsu użytkownika (widoku) przy użyciu dodatkowej warstwy (koordynatora). W przypadku aplikacji *Sandstorm* modelem jest klasa **Drone** z pakietu **hive**, widok reprezentuje klasa **MainFrame** (pakiet **ui**), natomiast rolę kontrolera spełnia **ClientController** (również z pakietu **ui**).

Dodatkowo pakiet **ui** zawiera wewnętrzny pakiet **dialogs**, w którym znajdują się moduły do obsługi okienek dialogowych.

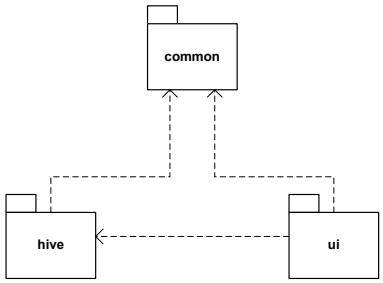
Działanie klas interfejsu użytkownika opiera się na bibliotece *wxPython* [37], która zapewnia wygodny, jednolity interfejs tworzenia aplikacji okienkowych dla różnych systemów operacyjnych. Implementacja własnego interfejsu odbywa się przez dziedziczenie ze zdefiniowanych w bibliotece klas, takich jak **wxFrame** (okno aplikacji), **wxDialog** (okienko dialogowe) oraz wykorzystywanie gotowych elementów: **wxButton** (przycisk), **wxTextCtrl** (pole do wpisywania tekstu) itp.

Pakiet common

Pakiet **common** zawiera moduły implementujące dodatkową funkcjonalność, która może być wykorzystywana przez inne pakiety. W skład pakietu **common** wchodzą m.in.: moduł algorytmu uzgadniania klucza Diffiego-Hellmana, moduł do generowania *unikatowych identyfikatorów*, moduł wypisywania informacji pomocniczych oraz moduł z niezmienialnymi ustawieniami programu.

Zależności

Rysunek 3.6 opisuje zależności między pakietami.



Rysunek 3.6: Diagram zależności między pakietami programu *Sandstorm*.

Jak widać, pakiet silnika **hive** nie zależy od pakietu interfejsu graficznego **ui** – oznacza to, że można zastosować zupełnie inny interfejs graficzny dla aplikacji bez zmieniania jej silnika.

3.4.2. Podział na wątki

Aby aplikacja sieciowa miała pożądaną wydajność, musi być w stanie obsługiwać wiele zapytań i zdarzeń jednocześnie i niezależnie od siebie. Z tego powodu aplikacja *Sandstorm* uruchamia kilka wątków bazowych oraz dodatkowe wątki, które uruchamiane są w miarę potrzeb.

W języku *Python* tworzenie wątku polega na utworzeniu obiektu klasy, która dziedziczy po klasie **threading.Thread**. Klasy wątków w projekcie *Sandstorm* to:

- **AnonymizerThread** – jego zadaniem jest uruchamianie w regularnych odstępach czasu procedur sprawdzających, czy struktury tuneli i sąsiadów są nadal aktywne i odbudowywanie ich w miarę potrzeb.
- **DroneDaemonThread** – wątek, na którym uruchamiany jest serwer *Python Remote Objects*, który oczekuje na żądania innych węzłów w sieci. Dla każdego żądania odebranego przez ten wątek, tworzony jest osobny, nowy wątek, który trwa do czasu obsłużenia żądania i jest potem niszczony.
- **DownloaderThread** – wątek tworzony dynamicznie; istnieje jeden taki wątek na każdy plik, który jest w trakcie pobierania. **DownloaderThread** zajmuje się koordynacją pobierania pliku – w tym celu co jakiś czas wysyła zapytania przeszukujące sieć w poszukiwaniu tego pliku, a także tworzy wątki pobierające części pliku (**PartThread**).
- **PartThread** – istnieje jeden taki wątek na każdego znalezionego właściciela każdego pobieranego pliku. Zadaniem tego wątku jest pobranie jednej części pliku od konkretnego właściciela i zapisanie jej na dysk. Wątki te są tworzone przez wątek **DownloaderThread**, gdy w sieci pojawią się nowi właściciele pliku.
- **FileSearchThread** – wątek służący do wyszukiwania pliku w sieci. Jego zadanie polega na sprawdzeniu, czy szukanego pliku nie ma wśród udostępnianych plików oraz zapytaniu wszystkich sąsiadów o ten plik.

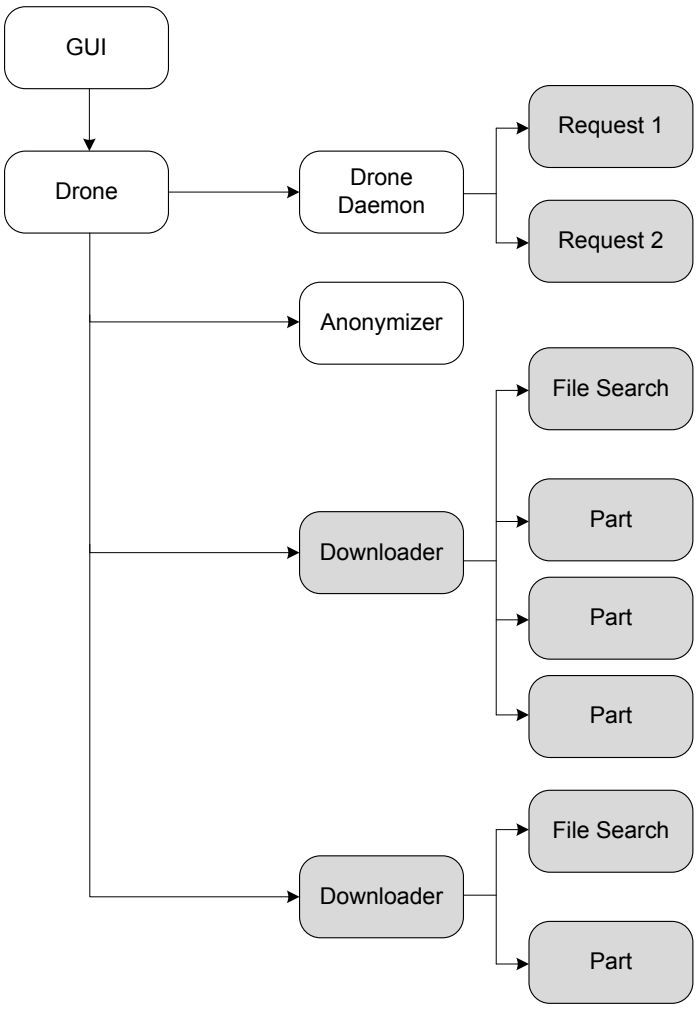
Rysunek 3.7 przedstawia przykładową sytuację w aplikacji z punktu widzenia podziału na wątki.

W przedstawionej sytuacji na stałe pracują wątki: interfejsu użytkownika (**GUI**), silnika (**Drone**), przetwarzania żądań (**Drone Daemon**) oraz **Anonymizer**. Wątek **Drone Daemon** przyjął dwa żądania od innych węzłów i jest w trakcie ich przetwarzania – dzieje się to w wątkach **Request 1** i **Request 2**. Dodatkowo pobierane są dwa pliki – dla każdego stworzony został wątek **Downloader**. Pierwszy plik jest pobierany z trzech źródeł jednocześnie, a drugi plik z jednego źródła (wątki **Part**). Dodatkowo dla każdego z plików uruchomione jest zadanie przeszukiwania sieci w poszukiwaniu nowych właścicieli (wątki **File Search**).

Jak widać, w aplikacji *Sandstorm*, poza głównym wątkiem i wątkiem interfejsu użytkownika, może działać jednocześnie od kilku do kilkudziesięciu wątków pomocniczych. Ponieważ wiele z nich korzysta z tych samych danych, muszą być one chronione przez jednoczesnym dostępem za pomocą mechanizmów wzajemnego wykluczania, takich jak kolejki i muteksy.

3.5. Anonimowość

W tym rozdziale opiszemy, w jakim stopniu aplikacja *Sandstorm* i jej podobne zapewniają anonimowość. Przyjrzymy się różnego rodzaju atakom mającym na celu złamanie tej anonimowości i sprawdzimy, jak można sobie z nimi radzić.



Rysunek 3.7: Podział aplikacji *Sandstorm* na wątki. Białym kolorem oznaczono wątki trwałe, szarym – wątki tworzone w razie potrzeby. Powiązania wskazują, kto sprawuje kontrolę nad poszczególnymi wątkami.

3.5.1. Stopnie anonimowości

Przez stopień anonimowości rozumiemy wysiłek, jaki musiałaby podjąć osoba chcąca poznać informacje, które chcemy przed nią ukryć, takie jak tożsamość, miejsce zamieszkania, to, z kim i kiedy się komunikujemy i jakie informacje wymieniamy.

W sensie teoretycznym idealna anonimowość nie istnieje – przy odpowiednich środkach i wystarczającej dozie cierpliwości zawsze można ją złamać. Sieci anonimowe takie jak *Sandstorm* opierają się na założeniu, że zużycie takich środków będzie dla atakującego nieopłacalne w porównaniu do korzyści, jakie by dzięki nim odniósł.

3.5.2. Charakterystyka sieci Sandstorm

Przypomnijmy najważniejsze cechy, które wpływają na zapewnienie anonimowości w sieci *Sandstorm*:

- Tworzenie tuneli dla użytkowników udostępniających treści. Węzeł, który tworzy tunel, wybiera tylko pierwszego pośrednika – kolejni są tworzeni rekurencyjnie tak, jakby pierwszy pośrednik był właścicielem tunelu. Tunele są tworzone od nowa przy każdym podłączeniu do programu oraz są z pewnym prawdopodobieństwem zmieniane raz na jakiś czas.

- Każdy użytkownik może ustalić własną liczbę oraz długość tuneli, a także liczbę bezpośrednich sąsiadów.

- Sieć jest niezależna od innych sieci i nie komunikuje się z węzłami niepodłączonymi.

- Każdy węzeł jest niezależny od innych i ma te same uprawnienia, co wszystkie inne węzły – nie ma serwerów kontrolnych ani superwęzłów nadzorujących pracę sieci.

Należy jednak pamiętać, że wiele funkcji programu opiera się na założeniu, że u wszystkich użytkowników działa ten sam program, który prawidłowo realizuje polecenia, zgodnie z założeniami protokołów i algorytmów. Nic nie stoi na przeszkodzie, by atakujący (tak będziemy nazywać osobę, która chce złamać naszą anonimowość) napisał własny program, który podszywa się pod prawdziwą aplikację *Sandstorm*, ale realizuje pewne funkcjonalności w inny sposób.

3.5.3. Ataki na anonimowość

W tej części przedstawiamy różne rodzaje możliwych ataków, których celem jest złamanie anonimowości, czyli poznanie informacji, którą chcieliśmy ukryć przed innymi użytkownikami sieci. Większość definicji ataków pochodzi z [11]. Przy opisywaniu ataków weźmiemy pod uwagę przede wszystkim koszty ze strony atakującego, możliwą do uzyskania informację oraz jak aplikacje mogą się bronić przed takimi atakami.

Ataki typu „brutalna siła”

Ataki typu „brutalna siła” polegają na obserwacji wszystkich wiadomości, które przepływają w sieci i wnioskowaniu z nich, którędy i od kogo przechodzą fragmenty plików. Przeprowadzenie takiego ataku nie jest łatwe, gdyż nie wszystkie wiadomości zawierają fragmenty plików (niektóre służą jedynie do utrzymywania sieci w spójnym stanie), a poza tym każda komunikacja jest szyfrowana oddzielnym kluczem. Oznacza to, że ta sama wiadomość przesyłana od użytkownika A do użytkownika B będzie „z zewnątrz” wyglądać inaczej, niż przesyłana od użytkownika B do użytkownika C.

Jedynym sposobem, aby wyśledzić pewien pakiet danych, jest przesłanie naprawdę dużej ich ilości, a następnie obserwacja, u których użytkowników nagle nastąpił bardzo duży przepływ danych. Większość użytkowników nie powinna przejmować się tym atakiem, gdyż jego przeprowadzenie jest bardzo kosztowne i często nielegalne (wymaga aktywności, którą można by nazwać podsłuchem). Ochrona przed tym atakiem w sieci *Sandstorm* powinna polegać na ograniczeniu przez osobistą ścianę ogniową przepustowości łącza przydzielonego aplikacji.

Ataki czasowe

Atak czasowy opiera się na założeniu, że protokoły komunikacji są schematyczne – na wysłane zapytanie zawsze przychodzi odpowiedź, wymiana klucza to cztery komunikaty, przesłanie

pliku – sześć itp. Atakujący, który potrafi obserwować ruch w sieci (co już wymaga ogromnych możliwości), mógłby szukać nadawcy np. przez eliminację węzłów, przez które na pewno nie mogła w danym przedziale czasu przejść obserwowana wiadomość.

Protokoły w sieci *Sandstorm* faktycznie są dosyć schematyczne – jedynym wyjątkiem jest dynamiczność tworzona przez fakt, iż tunele są różnej, losowej długości. Rozwiązaniem, które może pomóc w przeciwdziałaniu takim atakom, jest sztuczne dodawanie losowych opóźnień do wysyłanych pakietów – można to uzyskać np. za pomocą wyspecjalizowanego rutera lub programu do kontroli przepustowości sieci.

Ataki „na przecięcie"

Wyobraźmy sobie sytuację, w której tylko jeden użytkownik w sieci udostępnia plik P. Aby dowiedzieć się, który jest to użytkownik, należy poprosić go o ten plik (sprawdzić, czy jest podłączony do sieci), a następnie zapamiętać adresy wszystkich węzłów w sieci. Kiedy następnym razem podłączymy się do sieci i plik P znów jest dostępny, to znaczy, że użytkownik posiadający ten plik należy do przecięcia zbiorów: poprzednich użytkowników i aktualnych użytkowników. Stosując taką operację przecięcia wielokrotnie, można bardzo znacząco zawęzić obszar poszukiwań.

W sieci *Sandstorm* ten atak ma niskie szanse powodzenia, jeśli sieć jest wystarczająco duża (pomijając fakt, iż jest on bardzo kosztowny dla dużych sieci). Wynika to z prostego faktu, że atakujący nie może być pewien, że istnieje tylko jedna kopia pliku P w sieci. Obserwowany za jednym razem właściciel pliku może do następnego razu rozłączyć się, a na jego miejsce pojawić się ktoś inny, kto również posiada plik P. Atak „na przecięcie" wykluczyłby natychmiast obu właścicieli pliku, tymczasem niewinny użytkownik, który akurat był podłączony do sieci w obu przypadkach, zostałby uznany za podejrzanego.

Ataki odmowy dostępu

Atak odmowy dostępu polega na zablokowaniu możliwości komunikacji z innymi węzłami pewnemu węzłowi przez nieustanne wysyłanie do niego zapytań, na które musi on odpowiedzieć. Mogą to być zapytania typu *ping*, czyli sprawdzające, czy węzeł jest podłączony do sieci, ale także dowolne inne zapytania. Szczególnie groźne są polecenia wymagające od węzła intensywnego zużycia procesora, np. prośba o zaszyfrowane przesłanie dużego fragmentu pliku.

W sieci *Sandstorm* wszystkie węzły mają te same uprawnienia, a sama sieć jest odporna na awarię dowolnego węzła. Z tego powodu przeprowadzanie ataków odmowy dostępu, jeśli nie jest skierowane do znacznej (procentowo) liczby węzłów sieci jednocześnie, po prostu nic nie daje. Z kolei zablokowanie więcej niż połowy węzłów jednocześnie będzie miało bardzo negatywny wpływ na spójność sieci, ale znowu uważane jest za zbyt kosztowne, by było opłacalne.

Ataki przez podpisanie

Jeśli atakujący jest właścicielem węzła będącego pośrednikiem w czyimś tunelu, może dopisać pewien fragment do przekazywanej wiadomości tak, aby inny węzeł będący własnością atakującego mógł ten fragment odczytać i zorientować się, skąd nadeszła wiadomość. W ten sposób na przykład dwa węzły mogłyby zorientować się, że są w tym samym tunelu, a więc dalszy węzeł dowiedziałby się, którędy prowadzi droga do właściciela tunelu. Jest to jednak jedyna informacja, jaką mógłby uzyskać – nawet gdyby wszystkie węzły, z których składa się tunel, należały do atakującego, nie wiedziałby on, czy poprzednik pierwszego węzła tunelu jest już właścicielem, czy jedynie kolejnym pośrednikiem.

Ponieważ praktyczne ograniczenia na długość tunelu są bardzo duże (rzędu dziesiątek węzłów), ustalenie maksymalnej długości tunelu dla konkretnego węzła jest niemożliwe. Jedyną szansą atakującego jest możliwość, że użytkownik nie zmienił domyślnych ustawień zakresu długości tunelu: jeśli atakujący wie, że ma tunel o długości 5 oraz że poszukiwany węzeł tworzy tunele o długości od 3 do 5, to od razu widzi, że poprzednikiem pierwszego pośrednika w tym tunelu jest jego właściciel. Z tego powodu zalecane jest, by zawsze zmieniać domyślne ustawienia o długości tuneli.

Ataki dzielące

Atak dzielący *(ang. partitioning attack)* polega na obserwacji, jak zmieni się ruch w sieci, gdy zablokujemy te węzły, których nieobecność rozspójni sieć, podzieli ją na kilka części. Takie ataki mogą dać bardzo dużo informacji: jeśli użytkownik pobierał plik i po podziale nagle już nie może go pobrać, to oznacza, że nadawca był w części sieci, która została odcięta.

Ataki dzielące wymagają ogromnych możliwości ze strony atakującego – musi on być w stanie obserwować ruch w sieci oraz mieć możliwość wyłączania (np. przez atak odmowy dostępu) węzłów spajających sieć. Z tego powodu zagrożenie ze strony tych ataków jest często pomijane. *Sandstorm* próbuje bronić się przed tymi atakami: każdy węzeł posiada kilku sąsiadów, którzy są często weryfikowani (sprawdzane jest, czy nie rozłączyli się) i wymieniani na nowych. W ten sposób atakującemu trudno jest rozspójnić sieć.

Ataki „na poprzednika"

Schemat działania atakującego może być następujący: podłączamy wiele węzłów do sieci, te węzły stają się pośrednikami w czyichś tunelach, a następnie wysyłamy zapytanie o pewien plik i patrzymy, przez które z naszych węzłów to zapytanie przechodzi i od jakich (nie naszych) węzłów przychodzi przez tunel odpowiedź na to zapytanie. Po pewnym czasie okaże się, że niektóre węzły występują częściej niż inne i to wśród nich należy szukać właściciela pliku.

Ten atak wciąż wymaga sporo możliwości ze strony atakującego, choć nie tak wiele, jak ataki wymienione wcześniej. Nie daje on jednak wiele; to, że węzeł występuje częściej jako poprzednik wcale nie musi oznaczać, że jest on właścicielem – może on zawsze *wiarygodnie zaprzeczyć* i stwierdzić, że był jedynie kolejnym pośrednikiem.

Ataki typu Sybil

Kategoria ataków typu *Sybil* [7] opiera się na bardzo prostym pomyśle zbierania statystyk z sieci przez podłączenie do niej ogromnej liczby własnych węzłów. Wymaga to bardzo dużych możliwości atakującego, ale pozwala dowiedzieć się praktycznie wszystkiego – jeśli węzły atakującego stanowią ponad 90% wszystkich węzłów w sieci, anonimowość pozostałych praktycznie przestaje istnieć.

Sposobem zapobiegania takim atakom jest tzw. „płacenie za tożsamość", czyli ograniczanie podłączania nowych użytkowników przez nakładanie na nich różnych kar, np. finansowych albo czasowych. *Sandstorm* nie posiada zabezpieczeń przed atakami typu *Sybil*.

Ataki kryptograficzne

Aplikacja *Sandstorm* do zapewnienia prywatności używa szyfrowania: negocjacja klucza odbywa się przez algorytm Diffiego-Hellmana z 1024-bitowym rozmiarem grupy, do szyfrowania pakietów stosowany jest algorytm AES z 256-bitowym kluczem. Anonimowość opiera się

na bezpieczeństwie tych algorytmów. Gdyby którykolwiek z tych algorytmów został złamany, złamana byłaby również anonimowość aplikacji. Dotychczasowe doświadczenia i badania [3, 4, 14] pozwalają jednak mieć nadzieję, że tak się nie stanie.

Ataki na implementację

Przy tworzeniu skomplikowanych, współbieżnych i rozproszonych aplikacji takich jak *Sandstorm* możliwe jest, że pojawią się błędy w implementacji. Nawet jeśli aplikacja sama w sobie błędów nie zawiera, to opiera się na działaniu bibliotek: *Pyro* do komunikacji między węzłami, *PyCrypto* do szyfrowania oraz samego interpretera języka *Python*, które również mogą mieć nieznane luki w bezpieczeństwie. Atakujący może próbować znaleźć te błędy i wykorzystać je do różnych celów. W przypadku niektórych błędów, takich jak błędy przepełnienie bufora w programie interpretera języka Python, możliwe jest nawet przejęcie kontroli nad komputerem ofiary, co jest dużo gorszym efektem niż utrata anonimowości.

Rozdział 4

Testy wydajności

W celu zmierzenia wydajności aplikacji i sieci *Sandstorm* przeprowadzono szereg testów. Testy przeprowadzane były na komputerach PC z systemem *Microsoft Windows XP Professional*, połączonych siecią lokalną Ethernet o przepustowości 100Mbit/sek.

W sieci znajdowało się 9 komputerów, połączonych z losowymi sąsiadami. Na każdym komputerze uruchomione były dwie aplikacje klienckie. Ustawienia były następujące: pożądana liczba tuneli: 1, pożądana liczba sąsiadów: 5. Długość tuneli była ustalana w zależności od testu.

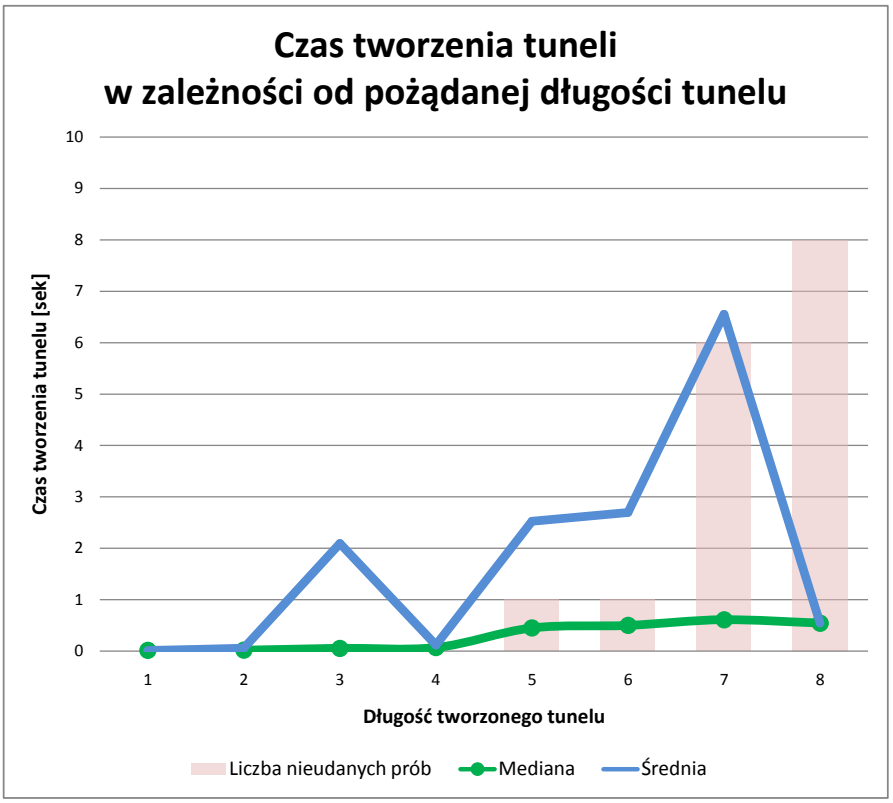
4.1. Tworzenie tunelu

Na początek przeprowadzono pomiary czasu tworzenia tuneli. Wykonano testy dla długości tunelu od 1 do 8, po 10 testów w serii dla każdej długości. Wyniki testów przedstawia tabela 4.1 oraz wykres 4.2.

Długość tunelu	1	2	3	4	5	6	7	8
Próba 1	0,02	0,02	0,03	0,03	0,45	0,48		0,25
Próba 2	0,00	0,03	0,11	0,38	0,14	0,64	0,13	0,84
Próba 3	0,02	0,02	10,19	0,33	10,08	20,09		
Próba 4	0,00	0,08	0,30	0,03	0,81	0,94	0,34	
Próba 5	0,00	0,02	10,08	0,05	0,11	0,19	0,61	
Próba 6	0,01	0,02	0,03	0,08	10,48	0,39		
Próba 7	0,00	0,02	0,03	0,08	0,48	0,41		
Próba 8	0,02	0,02	0,06	0,08		0,50		
Próba 9	0,02	0,31	0,05	0,06	0,11	0,61	11,55	
Próba 10	0,02	0,02	0,05	0,06	0,08		20,14	
Mediana	0,02	0,02	0,06	0,07	0,45	0,50	0,61	0,55
Średnia	0,01	0,06	2,09	0,12	2,53	2,69	6,55	0,55
Liczba nieudanych prób	0	0	0	0	1	1	6	8

Rysunek 4.1: Wyniki pomiarów czasu tworzenia tuneli (w sekundach). Puste pole oznacza, że tworzenie tunelu nie powiodło się.

Widać, że czas tworzenia tunelu, choć liczony w dziesiątych częściach sekundy, rośnie proporcjonalnie do długości tworzonego tunelu. Ma to swoje uzasadnienie – klient wysyła



Rysunek 4.2: Wyniki pomiarów czasu tworzenia tuneli

prośbę o stworzenie tunelu do pierwszego sąsiada, a ten po prostu próbuje stworzyć tunel o jeden węzeł krótszy.

Duża rozbieżność między średnią a medianą prób wskazuje na to, że czasem zdarzały się bardzo długie okresy oczekiwania na stworzenie tunelu. Prawdopodobnie były one spowodowane błędami połączeń i ponownymi próbami połączenia przez bibliotekę *Pyro*.

Zwiększająca się liczba nieudanych prób stworzenia tunelu jest efektem zbliżania się do granicy rozmiaru sieci – w sieci składającej się z 18 węzłów rzadko można trafić na 7 lub 8 powiązanych kolejno sąsiadów.

4.2. Pobieranie pliku

W testach szybkości pobierania plików posłużono się trzema przykładowymi plikami o rozmiarach: 4,15MB, 7,01MB i 10,2MB. Każdy plik pobierany był trzykrotnie dla wszystkich możliwych kombinacji ustawień:

- długości tuneli węzłów udostępniających (od 1 do 5);
- liczby węzłów udostępniających plik (od 1 do 5).

Wyniki pomiarów przedstawia tabela 4.3.
W kolejnych podrozdziałach przyjrzymy się otrzymanym wynikom z różnych perspektyw.

Dł. Tuneli	Liczba udostępniających	Plik o rozmiarze 4,15MB			Plik o rozmiarze 7,01MB			Plik o rozmiarze 10,2MB		
		Próba 1	Próba 2	Próba 3	Próba 1	Próba 2	Próba 3	Próba 1	Próba 2	Próba 3
1	1	27,56	36,69	36,97	60,56	60,94	60,67	76,31	76,05	74,81
	2	19,86	20,97	21,63	29,48	30,33	29,41	41,42	35,94	49,73
	3	20,11	39,75	19,56	29,63	24,67	24,09	31,05	30,81	31,08
	4	14,17	14,69	13,39	21,14	19,08	20,45	30,36	28,38	30,59
	5	13,51	12,83	13,16	18,27	19,41	18,11	29,16	25,36	27,09
2	1	36,75	35,52	37,61	53,20	50,94	65,11	65,21	64,68	64,06
	2	28,84	25,75	26,50	42,80	40,22	43,83	45,52	45,50	45,22
	3	19,59	26,39	18,31	30,25	29,64	31,66	38,89	37,56	36,19
	4	18,38	17,33	17,50	28,22	29,63	32,13	39,16	41,44	42,64
	5	16,17	15,70	16,05	28,61	23,39	27,19	37,80	44,14	35,50
3	1	41,61	39,60	44,60	57,97	58,14	61,38	74,07	72,87	72,34
	2	38,48	29,56	32,51	44,55	47,79	63,02	62,30	57,02	64,25
	3	24,57	27,32	35,87	34,21	33,23	34,79	49,77	42,01	73,74
	4	42,26	42,10	29,17	89,49	58,77	88,89	73,61	52,57	42,29
	5	19,43	18,75	20,78	31,64	34,54	34,42	39,17	38,20	40,12
4	1	63,28	70,87	62,98	98,12	112,19	78,92	88,43	81,74	84,71
	2	23,69	24,09	49,86	53,92	65,21	51,65	73,60	76,06	90,37
	3	22,45	34,52	40,03	43,78	38,95	48,21	52,76	63,87	67,43
	4	21,40	37,01	23,25	39,54	32,23	33,43	47,26	40,29	40,12
	5	29,03	30,89	45,16	36,57	36,89	34,62	50,06	57,03	55,60
5	1	96,13	54,59	54,04	125,54	175,54	83,53	142,54	154,26	118,34
	2	59,26	64,86	59,42	87,89	117,93	89,78	128,48	131,51	130,82
	3	50,25	73,73	80,65	76,42	73,53	96,87	150,90	146,46	148,12
	4	55,26	45,70	76,86	87,70	74,25	109,70	112,03	143,17	89,65
	5	222,89	67,47	65,34	117,30	88,10	111,77	90,59	108,56	100,17

Rysunek 4.3: Wyniki pomiarów czasu pobierania plików w sekundach

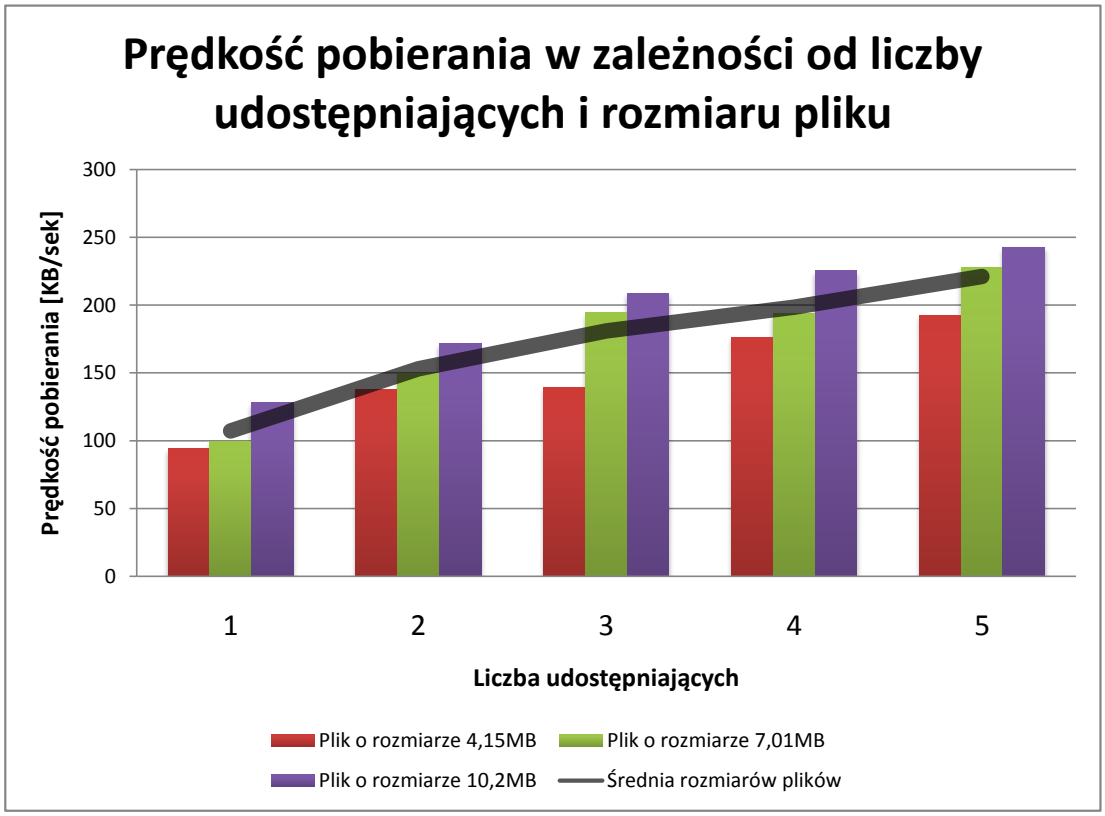
4.2.1. Prędkość w zależności od liczby udostępniających i rozmiaru pliku

Do testów zostały użyte trzy pliki o różnych rozmiarach. Aby łatwiej było porównywać otrzymane czasy pobierania, przy rozważaniu wydajności programu wygodnie będzie posługiwać się pojęciem prędkości pobierania pliku. Prędkość pobierania obliczamy, dzieląc rozmiar pliku przez czas jego pobrania. Jako jednostki prędkości używać będziemy kilobajtów na sekundę (KB/sek).

Wykres 4.4 przedstawia zależność prędkości pobierania pliku w zależności od liczby węzłów go udostępniających oraz rozmiaru pliku.

Widać, że średnia prędkość pobierania pliku rośnie wraz z rosnącą liczbą węzłów udostępniających – aplikacja jest w stanie pobierać dane od kilku węzłów jednocześnie. Największy zysk obserwujemy przy zwiększeniu liczby udostępniających z jednego do dwóch, potem wzrost prędkości jest już mniejszy. Spowodowane jest to koniecznością obsługi każdego udostępniającego przez pojedynczą aplikację klienta – musi on poświęcić czas na rozszyfrowanie pliku i zapisanie go na dysk.

Godny odnotowania jest fakt, iż dla większych plików osiągnięto większą średnią pręd-



Rysunek 4.4: Prędkość pobierania w zależności od liczby udostępniających i rozmiaru pliku

kość pobierania dla tej samej długości tunelu. Spowodowane jest to tym, że za każdym razem aplikacja testowa rozpoczynała pracę od sytuacji takiej, z jaką spotyka się użytkownik tuż po uruchomieniu programu – węzeł nie był połączony z żadnym innym. Ponieważ węzły wymieniają się losowo swoimi sąsiadami, to im dłużej węzeł jest podłączony do sieci, tym większa jest szansa na to, że jego sąsiedzi są „równomiernie rozłożeni", a więc klient ma dostęp do większej liczby potencjalnych posiadaczy pliku. Pobieranie większego pliku zabiera więcej czasu, stąd różnica w średniej prędkości jego pobierania.

4.2.2. Prędkość w zależności od długości tuneli i rozmiaru pliku

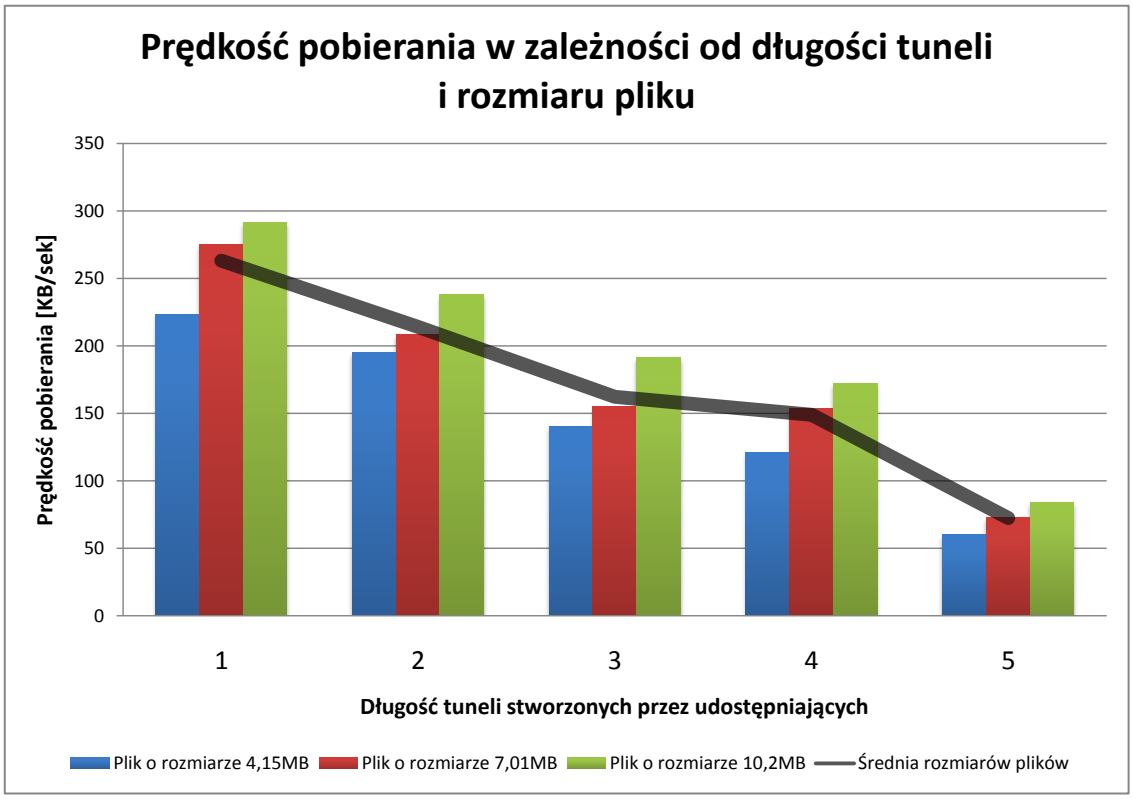
Na wykresie 4.5 przedstawiono zależność prędkości pobierania pliku w zależności od długości tuneli, jakie stworzyli dla siebie udostępniający, oraz od rozmiaru pliku.

Obserwujemy tu wyraźny spadek prędkości pobierania pliku wraz ze wzrostem długości tuneli węzłów go udostępniających. Spowodowane jest to dwoma czynnikami:

1. paczka z plikiem jest przesyłana przez sieć tyle razy, ilu jest pośredników w tunelu;
2. paczka z plikiem jest, oprócz szyfrowania między udostępniającym a odbiorcą, szyfrowana, a następnie rozszyfrowywana, tyle razy, ilu jest pośredników w tunelu.

Z tego powodu prędkość pobierania spada liniowo proporcjonalnie do długości tuneli węzłów udostępniających.

Ponownie obserwujemy, że większe pliki mają większą średnią prędkość pobierania – wyjaśnienie tego faktu jest takie samo, jak w rozdziale 4.2.1.



Rysunek 4.5: Prędkość pobierania w zależności od długości tuneli i rozmiaru pliku

4.2.3. Prędkość w zależności od liczby udostępniających i długości tuneli

Wykresy 4.6 i 4.7 prezentują zależność między liczbą udostępniających i długością tuneli a prędkością pobierania pliku.

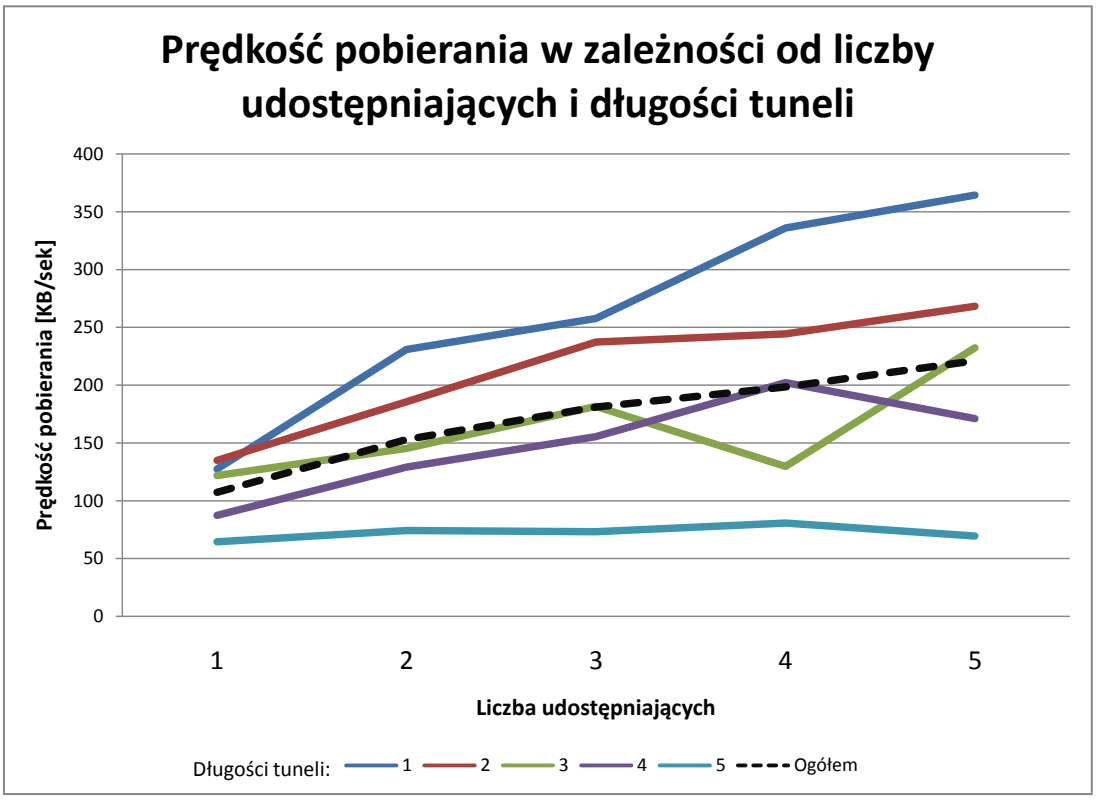
Na wykresie 4.6 uwydatniono zależność prędkości pobierania od liczby udostępniających.

Średnia prędkość pobierania pliku rośnie wraz ze wzrostem liczby posiadających go węzłów, ale wzrost ten jest różny dla różnych długości tuneli. Zauważamy, że im krótsze są tunele, tym prędkość przy dodawaniu kolejnych węzłów udostępniających rośnie szybciej – najszybciej prędkość pobierania rośnie dla tuneli o długości 1, a najwolniej dla tuneli o długości 5. Wynika to stąd, że im dłuższe są tunele, tym większy jest narzut dla udostępniających i pośredników przy wysyłaniu pliku. Zwiększa się czas od wysłania prośby o każdą część pliku do jego otrzymania, co opóźnia wysłanie prośby o kolejną część do tego samego węzła (na raz prosimy jeden węzeł tylko o jedną część).

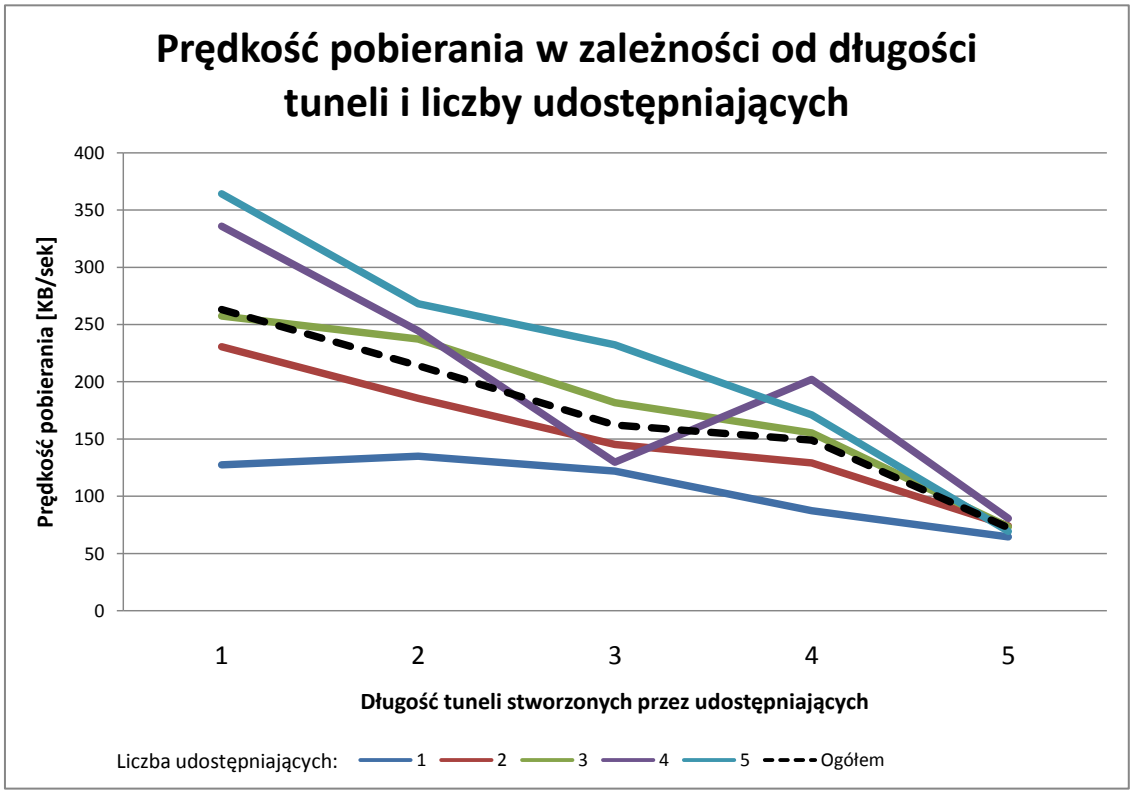
Przecięcie się wykresów dla tuneli o długości 3 i 4 przy czterech udostępniających należy uznać za losową kumulację błędów pomiaru – możliwe, że np. istotny dla pobierania pliku węzeł samoczynnie zmienił sąsiada i na jakiś czas przestał udostępniać dane.

Wykres 4.7 prezentuje tę samą zależność od drugiej strony – przedstawia, jak zmienia się prędkość pobierania przy zmianie długości tunelu.

Średnia prędkość pobierania pliku maleje wraz ze wzrostem długości tuneli tworzonych przez posiadające go węzły. Zauważmy, że im większa liczba udostępniających, tym szybciej prędkość maleje przy zwiększaniu długości tuneli. Największy spadek prędkości obserwujemy dla pięciu udostępniających, a najmniejszy – dla jednego.



Rysunek 4.6: Prędkość pobierania w zależności od liczby udostępniających i długości tuneli



Rysunek 4.7: Prędkość pobierania w zależności od długości tuneli i liczby udostępniających

4.3. Wnioski

Z przeprowadzonych testów wyraźnie wynika, że na prędkość pobierania plików wpływają dwa główne czynniki: liczba użytkowników posiadających i udostępniających plik oraz ich konfiguracja tuneli.

Pierwszy czynnik, liczba osób udostępniających pliki, wpływa dodatnio na prędkość pobierania dzięki współbieżności obsługi połączeń przez aplikację. Nie jest to czynnik charakterystyczny dla sieci *Sandstorm* – właściwie wszystkie sieci punkt-do-punktu wspomagają jednoczesne pobieranie treści od wielu użytkowników.

Konfiguracja tuneli natomiast jest czynnikiem, który każdy użytkownik musi sam ustalić, aby określić, jak istotna jest dla niego anonimowość, a jak wydajność. Chociaż tworzenie długich tuneli jest bardzo szybkie, to jednak koszt zwiększenia anonimowości nie jest niski – tracimy ok. 20% prędkości pobierania plików przy każdym zwiększeniu długości tunelu o 1. Dodatkowo przy dużych długościach tunelu, tj. zbliżających się do rozmiarów sieci lub większych niż ustalona w konfiguracji liczba sąsiadów, występują problemy z ich tworzeniem.

Rozdział 5

Podsumowanie

5.1. Możliwe rozszerzenia

Aplikacja *Sandstorm* jest w pełni sprawnym, działającym i odpornym na błędy programem, który w aktualnej postaci nadaje się do udostępnienia użytkownikom. W całości implementuje ona pomysł sieci, która zapewnia anonimowość jedynie użytkownikom udostępniającym treści. Aby jednak mogła ona konkurować z najpopularniejszymi sieciami nieanonimowymi, takimi jak *eDonkey2000* czy *BitTorrent*, potrzebne byłoby dodanie kilku usprawnień, które w aktualnej wersji zdecydowano się pominąć.

Przede wszystkim brakującym elementem jest własny system kolejkowania zapytań do węzła – aktualnie opiera się on na ograniczeniach biblioteki *Python Remote Objects* oraz systemu operacyjnego (*Microsoft Windows XP* z dodatkiem *Service Pack 2* pozwala na 10 półotwartych połączeń sieciowych jednocześnie). Zapytania, które nie mieszczą się w limicie, są odrzucane przez aplikację. Można zaimplementować system kolejkowania, który umieszczałby węzły w kolejce do obsługi, a następnie realizował żądania w kolejności ich przychodzenia. W ten sposób uniknęłoby się odrzucania żądań nadmiarowych. Należałoby jednak wtedy zastanowić się, co zrobić z prośbami o części pliku, które najpierw zostały wysłane i umieszczone w kolejce, a następnie pobierający zrezygnował i zdecydował się pobrać daną część od innego węzła. Aby rozwiązać ten problem, węzeł udostępniający mógłby wysyłać pytanie o potwierdzenie przed bezpośrednią realizacją każdego żądania w kolejce.

Używanie do obsługi zdalnych połączeń biblioteki *Python Remote Objects* ma wiele zalet, m.in. fakt, że nie trzeba samemu oprogramowywać gniazd sieciowych i rozpoznawać awarii. Jednak przy dużej skali sieci biblioteka ta może okazać się zbyt wolna i w rezultacie niewystarczająca. Można napisać własną wersję tej biblioteki, która posiadałaby mniejszą, ale wystarczającą funkcjonalność, a przy tym działała szybciej. Dodatkowo przydatna byłaby funkcja wstępnego odrzucania żądań – w tej chwili wszystkie prośby o połączenie, o ile są wolne łącza sieciowe, są przyjmowane. Złośliwy użytkownik mógłby napisać aplikację, która wysyła dużo żądań do węzła i w rezultacie wykluczyć go z użycia (patrz rozdział 3.5.3). Przydatny byłby wstępny etap, na którym biblioteka odrzucałaby takie nadmiarowe żądania. Inną funkcjonalnością, której brakuje bibliotece *Python Remote Objects*, jest możliwość omijania ścian ogniowych *(ang. firewall)* i ruterów – w tej chwili, aby komunikacja między węzłami mogła dojść do skutku, oba węzły muszą być osiągalne przez swój adres IP, co oznacza, że jeśli są podłączone do ruterów lub ścian ogniowych, to wymagają dodatkowej konfiguracji.

Istniejący graficzny interfejs użytkownika wystarcza do poprawnego funkcjonowania i wykorzystania wszystkich własności silnika systemu. Brakuje mu jednak drobnych udogodnień, które ułatwiają pracę, takich jak graficzne obrazowania drzewa udostępnianych plików, lista

ostatnich połączeń itp. Przydatna byłaby również funkcja zapamiętywania, w którym miejscu przerwało się pobieranie pliku przy wyłączaniu aplikacji – w tej chwili informacje te są tracone. Dodatkowo brakuje takich funkcji, jak automatyczna aktualizacja do najnowszej wersji programu czy interaktywna pomoc.

Sandstorm, mimo iż tworzony z myślą o systemie operacyjnym *Microsoft Windows XP*, został napisany w języku programowania, którego interpreter ma implementacje na wielu systemach operacyjnych. Również biblioteki, z których korzysta, są uniwersalne – można więc prawdopodobnie, z bardzo niewielkimi poprawkami, uruchomić go na innych systemach operacyjnych, takich jak np. *GNU/Linux*.

W kierunku samej anonimowości niewiele więcej da się zrobić. Można, tak jak jest to zrobione w *I2P*, dodać losowe opóźnienia przy wysyłaniu pakietów. Można łączyć pojedyncze wiadomości wysyłane do tego samego użytkownika w strumienie, aby obserwator nie mógł rozróżnić dwóch wiadomości od jednej. Zamiast szyfrowania symetrycznego warto rozważyć szyfrowanie asymetryczne (z użyciem dwóch kluczy: prywatnego i publicznego, patrz [14]) w celu zapewnienia większej integralności przesyłanych pakietów, będzie ono jednak zdecydowanie wolniejsze od szyfrowania symetrycznego.

W tej chwili użytkownik nie udostępnia pliku, dopóki go całego nie pobierze. Podyktowane jest to przede wszystkim chęcią zapewnienia anonimowości – wszak kiedy użytkownik pobiera treść, widoczny jest jego adres IP. Gdyby w tym samym momencie udostępniał, można by to było łatwo wykryć. Jednakże, przy zachowaniu odpowiednio dużych środków ostrożności, można by pokusić się o zaimplementowanie jednoczesnego pobierania i udostępniania. Pomysły, które się nasuwają, to: udostępnianie losowego podzbioru ukończonych części, udostępnianie tylko, gdy nie jest się jedynym pobierającym, prezentowanie każdemu pytającemu innego podzbioru posiadanych części. Przy implementacji tych pomysłów należy jednak bardzo uważać, by nie stać się podatnym na nowe ataki i nie utracić anonimowości.

5.2. Zakończenie

W pracy przedstawiono i przeanalizowano możliwe sposoby zapewniania użytkownikom anonimowości w Internecie. Dokonano przeglądu istniejących i używanych sieci anonimowych oraz sieci nieanonimowych, służących do wymiany plików między użytkownikami. Najlepsze cechy obu tych sieci, wraz z koncepcją połowicznej anonimowości, zapewniającej ochronę jedynie dla osób publikujących treści, stworzyły podstawę do zaprojektowania nowej sieci anonimowej.

Projekt nowej sieci, nazwanej *Sandstorm*, został następnie dokładnie opisany. Wymieniono algorytmy i protokoły, których używa, wskazano ich zalety i wady. Następnie opisano realizację pomysłu – aplikację stworzoną w języku programowania *Python*, która korzysta z sieci *Sandstorm* do wygodnej wymiany plików, zapewniającej anonimowość osób udostępniających pliki. Przeprowadzono dokładne testy wydajności sieci i aplikacji w działaniu na wielu komputerach.

Początkowy pomysł, by dla zapewnienia anonimowości użyć specyficznego charakteru protokołu *UDP* (możliwości niewypełnienia pola nadawcy w pakiecie *IP*), został zarzucony, gdy okazało się, że konieczność wypełnienia tego pola wprowadzili niezależnie od specyfikacji protokołu dostawcy Internetu w swoich ruterach. W założeniu miało to zapobiec rozproszonym atakom odmowy dostępu. W praktyce ataki te nadal są przeprowadzane, choć faktycznie potrzebna jest większa liczba komputerów, by odnieść ten sam skutek ataku, co przed wprowadzeniem ograniczeń.

Zdecydowano się więc użyć sprawdzonego rozwiązania, a mianowicie *tuneli*, czyli ciągu pośredników, przez których przechodzi wiadomość, zanim dotrze do adresata. W odróżnie-

niu jednak od tuneli stosowanych w istniejących sieciach anonimowych, w sieci *Sandstorm* zastosowano tunele wyłącznie dla węzłów publikujących treści.

Okazało się, że osłabienie warunku pełnej anonimowości (nadawcy i odbiorcy treści) do warunku anonimowości nadawcy stwarza możliwości około dwukrotnego zwiększenia wydajności sieci w porównaniu do sieci w pełni anonimowych. Testy wydajności wykazały, że straty wydajności spowodowane używaniem tuneli są znaczne, ale jest to cena, jaką płacimy za zapewnienie anonimowości. Pewien kompromis między anonimowością a prędkością można uzyskać, odpowiednio konfigurując długości używanych tuneli. Daje to nadzieję, że osłabianie innych warunków, takich jak siła anonimowości, spowoduje dalszy wzrost wydajności. Często bowiem nie zależy udostępniającemu na tym, żeby nikt nie wiedział, kim on jest, ale żeby nie potrafił tego w wiarygodny sposób udowodnić (wspomniana w pracy *wiarygodna zaprzeczalność*). Jest więc wiele możliwości rozwoju zarówno samej aplikacji *Sandstorm*, takich jak implementacja sprawniejszej biblioteki komunikacji między węzłami czy dalszych udogodnień dla użytkownika, jak i pomysłu sieci o niepełnej anonimowości.

Ciągłe powstawanie projektów sieci anonimowych, takich jak *Sandstorm*, oznacza, że ludziom zależy na przywróceniu anonimowości w sieci. Być może kiedyś, dzięki nim, internauci będą mogli cieszyć się taką samą wolnością słowa, jak w początkach działania Internetu.

Dodatek A

Instrukcja użytkownika programu Sandstorm

A.1. Zawartość płyty CD

Na płycie CD dołączonej do niniejszej pracy znajdują się następujące pliki:

- `Jancewicz-Anonimowa_publikacja_tresci_w_Internecie.pdf` – wersja pracy magisterskiej w formacie PDF;
- `Sandstorm.zip` – archiwum ZIP z programem *Sandstorm*;
- `wymagania/` – programy potrzebne do działania aplikacji *Sandstorm*:
  - `python-2.5.msi` – interpreter języka programowania *Python* w wersji 2.5 pod Windows;
  - `pycrypto-2.0.1.win32-py2.5.zip` – biblioteka kryptograficzna *Python Cryptography Tookit* dla *Pythona* 2.5 w wersji 2.0.1 pod Windows;
  - `wxPython2.8-win32-unicode-2.8.1.1-py25.exe` – biblioteka *wxPython* dla *Pythona* 2.5 w wersji 2.8.1.1 pod Windows;
  - `Pyro-3.6.win32.exe` – biblioteka *Python Remote Objects* w wersji 3.6 pod Windows.

A.2. Wymagania

Do poprawnego działania programu *Sandstorm* potrzebne są:

- system operacyjny *Microsoft Windows XP* lub nowszy;
- zainstalowany interpreter języka *Python* [33] w wersji 2.5 lub nowszej;
- zainstalowane moduły *Pythona*:
  - *Python Cryptography Toolkit* [32] w wersji 2.0.1 lub nowszej;
  - *Python Remote Objects* [34] w wersji 3.6 lub nowszej;
  - *wxPython* [37] (typ unicode) w wersji 2.8.1.1 lub nowszej.

Wszystkie wymienione moduły oraz interpreter Pythona można pobrać z Internetu bądź z płyty CD dołączonej do niniejszej pracy.

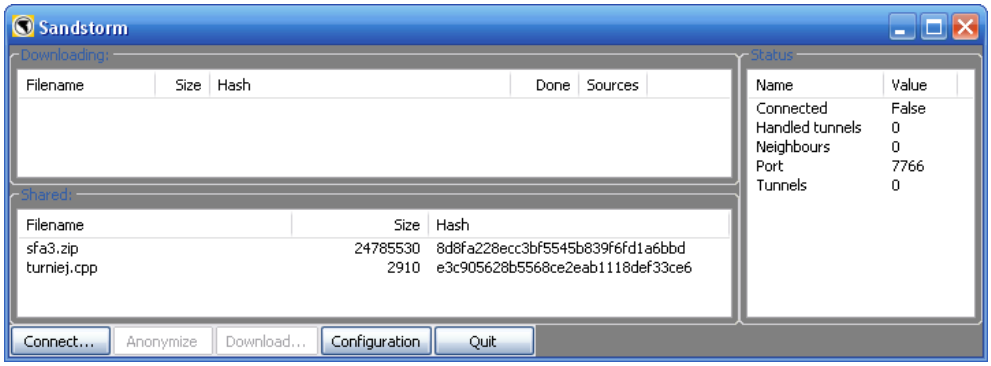
A.3. Instalacja

Aby zainstalować i uruchomić program *Sandstorm*, należy:

- zainstalować składniki, o których mowa w rozdziale A.2;
- rozpakować paczkę `Sandstorm.zip`, znajdującą się w głównym katalogu załączonej płyty CD, do dowolnego katalogu;
- uruchomić program przez wywołanie skryptu `Sandstorm.bat`, znajdującego się w głównym katalogu programu.

A.4. Używanie aplikacji

Po uruchomieniu aplikacji *Sandstorm* ukaże się główne okno programu (rysunek A.1).



Rysunek A.1: Główne okno programu Sandstorm

Ekran aplikacji jest podzielony na cztery części.

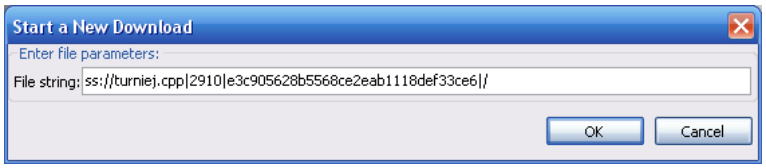
W lewej górnej sekcji znajduje się lista aktualnie pobieranych plików. Dla każdego pliku podane są informacje: nazwa, rozmiar, suma kontrolna, procent pobranych części oraz liczba węzłów w sieci, od których pobieramy plik.

Lewa dolna sekcja zawiera listę plików przez nas udostępnianych, wraz z ich rozmiarem oraz sumą kontrolną. Kliknięcie prawym przyciskiem myszy na udostępniany plik spowoduje skopiowanie jego *URI* do schowka systemu operacyjnego.

Prawa sekcja zawiera statystyki przydatne w pracy z programem: czy jesteśmy podłączeni do sieci, ilu mamy sąsiadów i tuneli, ile tuneli obsługujemy i na którym porcie działa aplikacja.

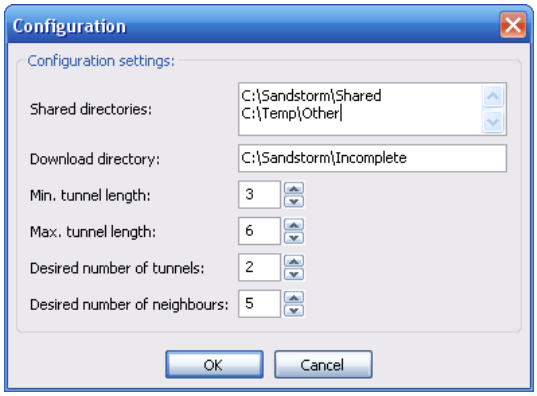
Opcje dostępne są w postaci przycisków znajdujących się w dolnej części okna:

- **Connect...** – pozwala na podanie nazwy komputera oraz portu, na którym działa inna aplikacja *Sandstorm*. Po wprowadzeniu informacji przez użytkownika następuje próba połączenia.
- **Anonymize** – rozpoczyna procedurę tworzenia tuneli w celu zapewnienia anonimowości. Aplikacja nie będzie w stanie wysyłać żadnych plików, jeśli nie będzie posiadała tuneli. Procedura tworzenia tuneli jest uruchamiana przez samą aplikację raz na jakiś czas – ten przycisk służy do wymuszenia jej uruchomienia.
- **Download...** – rozpoczyna pobieranie kolejnego pliku. Po wciśnięciu przycisku należy podać *URI* szukanego pliku (rysunek A.2).



Rysunek A.2: Okno pobierania nowego pliku w programie Sandstorm

- **Configuration** – umożliwia edycję konfiguracji. W konfiguracji zmienić można: katalogi z plikami, docelową liczbę sąsiadów w sieci oraz parametry tuneli (rysunek A.3).



Rysunek A.3: Okno konfiguracji programu Sandstorm

- **Quit** – kończy pracę z aplikacją.

Bibliografia

[1] I. Clarke, O. Sandberg, B. Wiley, T. W. Hong: *Freenet: A Distributed Anonymous Information Storage and Retrieval System*, 2000

[2] B. Cohen: *Incentives build robustness in BitTorrent*, Workshop on Economics of Peer-to-Peer Systems, Berkeley, CA, USA, 2003

[3] J. Daemen, V. Rijmen: *The Design of Rijndael: AES – The Advanced Encryption Standard*, Springer, 2002

[4] W. Diffie, M. E. Hellman, *New Directions in Cryptography*, IEEE Transactions on Information Theory, 1976

[5] R. Dingledine, N. Mathewson, P. Syverson: *Tor: The Second-Generation Onion Router*, Proceedings of the 13th USENIX Security Symposium, 2004

[6] M. Dorigo, T. Stützle: *Ant Colony Optimization*, MIT Press, 2004

[7] J. Douceur: *The Sybil Attack*, Proceedings of the 1st International Peer To Peer Systems Workshop, 2002

[8] M. Fowler: *Patterns of Enterprise Application Architecture*, Addison-Wesley, 2003

[9] M.Fowler: *UML Distilled: A Brief Guide to the Standard Object Modeling Language*, Addison-Wesley, 2003

[10] E. Gamma, R. Helm, R. Johnson, J. Vlissides: *Design Patterns: Elements of Reusable Object-Oriented Software*, Addison-Wesley, 1994

[11] J. Raymond: *Traffic Analysis: Protocols, Attacks, Design Issues and Open Problems*, Designing Privacy Enhancing Technologies: Proceedings of International Workshop on Design Issues in Anonymity and Unobservability, 2001

[12] R. Rivest: *The MD4 message-digest algorithm.* Request for Comments (RFC) 1320, Internet Activities Board, Internet Privacy Task Force, 1992

[13] R. Rivest: *The MD5 message-digest algorithm.* Request for Comments (RFC) 1321, Internet Activities Board, Internet Privacy Task Force, 1992

[14] B. Schneier: *Applied Cryptography, Second Edition*, Wiley & Sons, 1996

[15] A. S. Tanenbaum, *Computer Networks*, 4th Edition, Prentice-Hall International, Inc, 2002

[16] *Internet Protocol. DARPA Internet Program. Protocol Specification.* Request for Comments (RFC) 791, The Internet Engineering Task Force, 1981

[17] *Secure Hash Standard*, NIST, U.S. Department of Commerce, Federal Information Processing Standards Publication (FIPS PUB) 180-1, 1995

[18] *Transmission Control Protocol. DARPA Internet Program. Protocol Specification.* Request for Comments (RFC) 793, The Internet Engineering Task Force, 1981

[19] *Uniform Resource Identifiers (URI): Generic Syntax* Request for Comments (RFC) 2396, The Internet Engineering Task Force, 1998

[20] *User Datagram Protocol*, Request for Comments (RFC) 768, The Internet Engineering Task Force, 1980

[21] *Ustawa z dnia 4 lutego 1994 r. o prawie autorskim i prawach pokrewnych*, Dz.U. 1994 nr 24 poz. 83

[22] *ANA Spoofer Project*, `http://spoofer.csail.mit.edu/`

[23] *Azureus: Java Bittorrent Client*, `http://azureus.sourceforge.net/`

[24] *Eclipse - an open development platform*, `http://www.eclipse.org/`

[25] *eDonkey2000 Network*, Wikipedia, `http://en.wikipedia.org/wiki/EDonkey_network`

[26] *GNU General Public License*, `http://www.gnu.org/copyleft/gpl.html`

[27] *I2P*, `http://www.i2p.net/`

[28] *Official eMule Homepage*, `http://www.emule-project.net/`

[29] *Mute: Simple, Anonymous File Sharing*, `http://mute-net.sourceforge.net/`

[30] *PyDev: Python Development Environment*, `http://pydev.sourceforge.net/`

[31] *PyLint: Quality Assurance Module for Python*, `http://www.logilab.org/857`

[32] *Python Cryptography Toolkit*, `http://www.amk.ca/python/code/crypto.html`

[33] *Python Programming Language – Official Website*, `http://www.python.org/`

[34] *Python Remote Objects*, `http://pyro.sourceforge.net/`

[35] *The Freenet Project*, `http://freenetproject.org/`

[36] *Tor: anonymity online*, `http://tor.eff.org/`

[37] *wxPython: a GUI toolkit for Python*, `http://www.wxpython.org/`

[38] *µTorrent: Powerful BitTorrent Client*, `http://www.utorrent.com/`