Politechnika Wrocławska

Wydział Informatyki i Zarządzania
kierunek studiów: Informatyka
specjalność: Systemy Informacyjne

Praca dyplomowa - magisterska

Tworzenie ukrytych kanałów informacyjnych w sieciach komputerowych TCP/IP

Michał Rogala

http://www.michalrogala.com/security/whitenoise

słowa kluczowe:
ukryte kanały informacyjne
TCP/IP
covert channels

krótkie streszczenie:

Praca omawia teorię ukrytych kanałów informacyjnych oraz możliwości ich tworzenia we współczesnych sieciach komputerowych opartych o protokół TCP/IP. Przedstawione zostają istniejące już rozwiązania w tym zakresie oraz omówione ich wady mogące powodować błędy w transmisji oraz ograniczenia w rzeczywistych zastosowaniach. Postawionym w pracy problemem jest możliwość stworzenia oraz zaimplementowania protokołu realizującego w pełni niezawodny i niewykrywalny, dwukierunkowy, ukryty kanał informacyjny ukryty w pakietach protokołu TCP/IP. Stworzone rozwiązanie zostało przetestowane pod kątem niezawodności oraz efektywności prowadzenia transmisji. Wyznaczone zostały ograniczenia stworzonego protokołu.

Promotor:	Dr inż. Bogdan Trawiński	.....	.....
	imię i nazwisko	*ocena*	*podpis*

pieczątka Instytutu, w którym student wykonywał pracę

Wrocław 2008

Spis treści

Streszczenie pracy

W niniejszej pracy przedstawiona została koncepcja ukrytych kanałów informacyjnych w sieciach komputerowych opartych o protokół TCP/IP. Przedstawione zostały nieliczne badania i rozwiązania poświęcone temu zagadnieniu oraz przeanalizowane zostały ich wady uniemożliwiające zestawienie pełnego i niezawodnego kanału do wymiany danych. Problemem postawionym w niniejszej pracy jest możliwość zaprojektowania oraz zaimplementowania specjalistycznego protokołu służącego do wymiany danych w ukrytych kanałach informacyjnych. Protokołu zapewniającego funkcjonalności kanałów jawnych – dwukierunkowości transmisji, detekcji oraz korekcji błędów i równoległości transmisji. Głównym elementem pracy jest program w postaci sterownika sieciowego systemu Linux realizujący zaprojektowany protokół (nazwie roboczej Whitenoise) oraz umożliwiający aplikacjom zewnętrznym zestawianie kanałów komunikacyjnych. Zaimplementowane mechanizmy pozwoliły na przeprowadzenie badań niezawodności oraz efektywności zaprojektowanych rozwiązań.

Abstract

The subject of this thesis is an idea of creation of covert communication channels over the TCP/IP based computer networks. This work contains the analysis of known mechanisms of embedding hidden data into the TCP/IP packets and solutions or creating covert channels, developed so far. Due to many disadvantages that available algorithms have, they can't be used to create reliable information channels. The problem raised in present thesis is the possibility of designing a specialist protocol, used for reliable data transmission over covert channels. Such protocol should provide functionality of overt channels – bidirectional transmission, error detection and correction, but also a possibility of establishing parallel transmissions. The Linux kernel network module implementing such protocol (codename Whitenoise) and providing functionality for external userspace applications to establish covert channels is the main element of this work. Implemented mechanisms were submitted to research on reliability and effectiveness of designed solutions.

Wprowadzenie

Od roku 1972, który uznaje się za datę powstania Internetu, pojęcie informacji i ich wymiany diametralnie zmieniło swoje miejsce w cywilizowanym świecie, zarówno wśród pojedynczych osób jak i wielkiego biznesu. Lata 90 XX wieku oraz czas obecny, okrzyknięte erą informacji uświadomiły wszystkim, iż stwierdzenie *informacja jest potęgą* nie tylko nie traci na aktualności lecz z każdym kolejnym rokiem wkracza w coraz to nowe dziedziny naszego życia, sprawiając iż sieci komputerowe oplatają nas na każdym kroku.

Oparcie systemów wymiany informacji na sieci Internet, na przestrzeni lat zaczęło przynosić coraz więcej problemów i zagrożeń. Stos protokołów służących do wymiany danych w Internecie - TCP/IP stworzony w latach 70 XX wieku przez amerykańskie uniwersytety w Stanford, Utah oraz Los Angeles na potrzeby wojskowego projektu ARPANET, skupiał się głównie na zapewnieniu maksymalnej odporności na przekłamania transmisji oraz wykrywaniu i omijaniu uszkodzeń poszczególnych węzłów sieci. Niemal całkowicie jednak pomijał sprawy poufności przesyłanych danych oraz zabezpieczeń przed ich nieuprawnioną modyfikacją. Oceniając z dzisiejszego punktu widzenia proces projektowania TCP/IP należy jednak pamiętać o realiach tamtych czasów - nieliczne komputery dostępne w ośrodkach naukowych i wojskowych, bardzo powolne i zawodne łącza sieciowe oraz wymagania zleceniodawcy – armii USA, która przygotowując się w czasie zimnej wojny na konflikt z ZSRR szukała maksymalnie niezawodnego sposobu na komunikację poszczególnych ośrodków militarnych - nawet w warunkach wojny atomowej.

Otwarcie ARPANETU (nazywanego już Internetem) na liczne ośrodki naukowe na świecie oraz dynamiczny rozwój i popularyzacja sprzętu komputerowego w latach 80 XX w. sprawiła, iż do sieci zaczęła mieć dostęp coraz większa grupa osób – pojawiały się pierwsze zagrożenia informacyjne. Robak internetowy Roberta Morrisa, który sparaliżował w roku 1988 ponad 10% wszystkich komputerów podłączonych do sieci oraz próby nieautoryzowanego dostępu do komputerów sprawiły, iż pionierskie czasy w których każdy ufał każdemu szybko odeszły w niepamięć. Coraz większe ilości danych przesyłane przez Internet, nowe kraje podłączające się do sieci, oraz wzrost komercyjnego znaczenia informacji w formie elektronicznej sprawiły, iż pojawiły się grupy osób wykorzystujące słabości protokołów TCP/IP do naruszania poufności i integralności danych w systemach komputerowych - czy to dla zysku, czy dla zabawy. Tak też ujawniono zagrożenia ukryte w TCP/IP – możliwość podsłuchiwania bez większego problemu transmisji sieciowych - *sniffing*, podszywanie się pod innych użytkowników sieci - *spoofing*, czy też przechwytywanie i modyfikacja danych w autoryzowanych połączeniach innych osób - *session hijacking*.

Zastosowania Internetu w biznesie i do obsługi transakcji finansowych sprawiły, iż kwestie bezpieczeństwa przesyłanych informacji stały się przeliczalne na dolary i zwróciły uwagę na problemy w istniejących rozwiązaniach technicznych. Wprowadzenie kryptografii do połączeń internetowych (np. protokoły SSL i IPSec) oraz coraz bardziej restrykcyjne polityki bezpieczeństwa danych i monitoringu użytkowników, stosowane przez korporacyjnych użytkowników Sieci znacząco podniosły zaufanie do ich systemów. Równocześnie ze skupianiem dużej części obsługi ruchu internetowego w rękach firm i korporacji (często zależnych od rządów różnych krajów) w świecie współczesnej informatyki rozgorzały dyskusje dotyczące zachowania prywatności użytkowników Sieci oraz

przypadków cenzurowania całych segmentów Internetu ze względów politycznych i represji za korzystanie z *"zakazanych"* zasobów.

Badania nad prywatnością w Internecie oraz ochroną przed szpiegowstwem doprowadziły do powstania koncepcji *network steganography* - techniki ukrywania i niezauważonego przesyłania informacji w sieciach komputerowych. Techniki zdającej egzamin w warunkach, gdy nie można być pewnym skuteczności żadnego z zabezpieczeń komputerowych, a transmisje sieciowe, nawet szyfrowane, są narażone na podsłuch lub manipulację.

Celem niniejszej pracy dyplomowej jest zaprojektowanie i zaimplementowanie protokołu pozwalającego na tworzenie w pełni funkcjonalnych i niezawodnych, dwukierunkowych, ukrytych kanałów informacyjnych. Koncepcja obejmować będzie możliwości uniezależnienia algorytmów tworzenia takich kanałów od poszczególnych metod osadzania danych w ruchu sieciowym. Implementacja zaś, pod postacią modułu jądra systemu Linux, będzie otwartym środowiskiem pozwalającym na dokładne przebadanie użytych rozwiązań, a także łatwą ich modyfikację – umożliwiając w przyszłości dalsze prace nad zagadnieniem ukrytych kanałów informacyjnych.

W pierwszym rozdziale omówiona zostanie teoria zarówno jawnych, jak i ukrytych kanałów informacyjnych. Przedstawione zostaną również stawiane im wymagania.

W drugim rozdziale przybliżony zostanie sieciowy model OSI, a także jego dwa protokoły – TCP i IP - na których opierać się będą tworzone ukryte kanały.

W rozdziale trzecim omówione zostaną istniejące już wyniki badań, a także rozwiązania techniczne poświęcone tworzeniu ukrytych kanałów informacyjnych w TCP/IP.

Rozdział czwarty przedstawi zaprojektowany, autorski, protokół służący do zestawiania ukrytych kanałów – Whitenoise. Omówione zostaną dokładnie używane struktury danych, algorytmy działania oraz mechanizmy osadzania danych w pakietach sieciowych.

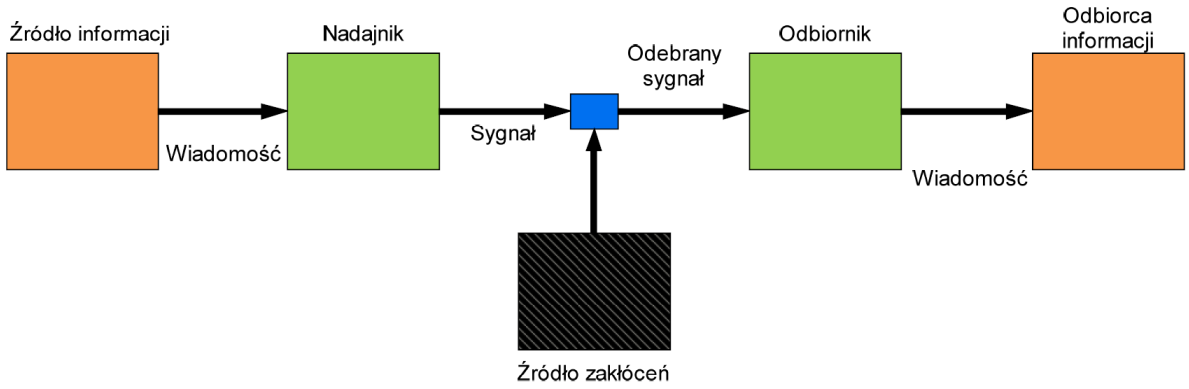
W rozdziale piątym przedstawione zostaną szczegóły techniczne implementacji protokołu Whitenoise w systemie Linux, a także interfejs programistyczny pozwalający na korzystanie z niego aplikacjom zewnętrznym.

Rozdział szósty przedstawi wyniki przeprowadzonych na protokole i implementacji testów poprawności oraz wydajności. Omówione również zostaną badania dotyczące wykrywalności ukrytych kanałów.

1 Jawne i ukryte kanały informacyjne

Analiza systemów komunikacyjnych oparta jest na abstrakcyjnym pojęciu kanału informacyjnego jako sposobie na przekazywanie wiadomości od nadawcy do odbiorcy za pomocą fizycznego medium. W ujęciu Shannonowskim [17] polega on na próbie reprodukcji (z ustalonym prawdopodobieństwem) po stronie odbiorcy tej wybranej przez nadawcę z określonego zbioru możliwych wiadomości.

Za system będący reprezentacją kanału informacyjnego uznamy więc układ przedstawiony na Rys. 1, składający się z 6 podstawowych elementów:



Rys. 1 Model kanału informacyjnego w ujęciu Shannonowskim

1. **Źródła wiadomości** – produkującego wiadomości lub sekwencje wiadomości z określonego zbioru.
2. **Nadajnika** – przekształcającego wiadomości ze źródła do postaci umożliwiającej przesłanie ich poprzez medium.
3. **Kanału** – medium fizycznego poprzez który wędrują sygnały do odbiorcy.
4. **Odbiornika** – przekształcającego sygnał z medium do postaci wiadomości (czyli realizującego pracę odwrotną do przekaźnika).
5. **Odbiorcy wiadomości** – miejsca/osoby/urządzenia dla którego przeznaczona jest dana wiadomość.
6. **Źródła zakłóceń** – wprowadzającego losowe zmiany w medium transmisyjnym – powodując możliwe błędne zdekodowanie wiadomości po stronie odbiorcy.

Zdefiniowany przez Shannona, przedstawiony powyżej model kanału informacyjnego nazywać będziemy kanałem typu *punkt-punkt* (lub zamiennie *unikastowym* lub *jeden do jednego*) gdyż wymiana informacji odbywa się naraz pomiędzy tylko dwoma uczestnikami (punktami), w przeciwieństwie do innych istniejących modeli – *jeden do wielu* (kanał rozgłoszeniowy) oraz *wielu do jednego*. Na potrzeby niniejszej pracy skupiać się będziemy jedynie na modelu *jeden do jednego* gdyż ten model jest podstawą komunikacji we współczesnych sieciach komputerowych.

Z każdym kanałem nierozerwalnie związanych jest szereg parametrów, które opisują jego właściwości [2]:

a) **Przepustowość** – określająca ilość informacji, które można przesłać poprzez kanał w danej jednostce czasu – zazwyczaj wyrażana w bitach na sekundę (bps).
b) **Niezawodność** – wyrażana za pomocą wartości BER (Bit Error Ratio) powstałej po podzieleniu liczby błędnie odebranych bitów informacji przez całkowitą liczbę odebranych bitów w kanale.

c) **Opóźnienie** – średni czas, który upływa od momentu przekazania wiadomości do przekaźnika przez nadawcę do momentu otrzymania wiadomości przez odbiorcę od odbiornika.
d) **Tryb transmisji** – synchroniczny lub asynchroniczny.

W świecie rzeczywistym niemożliwe jest stworzenie kanału informacyjnego, który cechowałby się stuprocentową niezawodnością w transmisji danych – wynika to głownie z konieczności istnienia medium fizycznego, które zawsze jest podatne na wpływ czynników zewnętrznych powodujących przekłamania. Dlatego też w trakcie projektowania protokołu transmisji danych niemal zawsze stosuje się mechanizmy pozwalające na wykrycie oraz korekcję błędów transmisji [2]: kodowanie nadmiarowe, sprawdzanie parzystości bitów, sum kontrolnych oraz strategie retransmisji danych (wymagające istnienia kanału komunikacji dwustronnej – w celu powiadomienia nadawcy o konieczności przesłania informacji ponownie).

Projektując model transmisji danych musimy brać także pod uwagę możliwość istnienia obok siebie wielu równolegle działających kanałów informacyjnych współdzielących to samo medium (np. kabel, częstotliwość radiową) i uwzględnienia aby nie kolidowały one ze sobą. Z zagadnieniem tym związane jest również pojęcie sesji [2] jako sparowanego połączenia pomiędzy nadawcą i odbiorcą. Używając analogii do rozmowy telefonicznej (gdzie dwie osoby mają cały czas dostęp do linii telefonicznej ale dzwonią do siebie raz na jakiś czas) nawiązanie sesji poprzedza wymianę danych (pozwala np. na synchronizację nadawcy i odbiorcy) oraz pozwala na jednoznaczną identyfikację (i/lub autoryzację) stron prowadzonej transmisji. Zamknięcie sesji kończy wymianę danych pomiędzy nadawcą i odbiorcą i zwalnia ich medium transmisyjne.

Podsumowując, chcąc zaprojektować niezawodny kanał wymiany danych trzeba wziąć pod uwagę następujące elementy:

1) Istnienie pary przekaźnik – odbiornik zajmujących się kodowaniem i dekodowaniem informacji oraz będących odpowiedzialnymi za ich transmisję poprzez medium. Ich działanie (np. sposób kodowania informacji) powinno być całkowicie przeźroczyste dla odbiorcy i nadawcy wiadomości (nie muszą znać szczegółów ich realizacji oraz pracy) .
2) Możliwość współdzielenia medium transmisyjnego przez wiele istniejących naraz kanałów.
3) Istnienie mechanizmu sesji – pozwalającego na jednoznaczną identyfikację nadawcy i odbiorcy oraz ustalenie momentu rozpoczęcia i zakończenia wymiany danych.
4) Istnienie mechanizmów wykrywających i neutralizujących skutki nieprawidłowej transmisji danych przez medium transmisyjne – detekcja i korekcja błędów.
5) Maksymalizację przepustowości kanału i minimalizację jego opóźnień.

1.2 Ukryte kanały informacyjne

Ukryte kanały informacyjne są dziedziną bezpieczeństwa informacji, która pojawia się w debatach środowisk akademickich, badawczych oraz rządowych od nieco ponad 30 lat. Po raz pierwszy pojęcie takiego kanału zostało zdefiniowane przez Lampsona [12] w roku 1973 jako "*metody transmisji informacji poprzez kanał lub medium nie zaprojektowane do*

komunikacji". Obecnie uznaje się bardziej dokładną definicję przyjętą przez Departament Obrony USA [4] jako *"dowolnego kanału komunikacji, który może posłużyć do transmisji informacji w sposób, który narusza politykę bezpieczeństwa danego systemu"* .

Ukryte kanały informacyjne dzieli się zasadniczo na:

a) **Kanały pamięciowe** – najprostsze i najpopularniejsze - w których dane przekazywane są poprzez zapisywanie ich w miejscach zasadniczo nie przeznaczonych do przechowywania informacji (np. miejsca na instrukcje sterujące, a nie dane).

b) **Kanały czasowe** – wykorzystujące do komunikacji średni czas potrzebny systemowi lub procesowi na wykonanie jakiejś operacji. Nadawca chcąc zasygnalizować jakąś wiadomość może obciążyć system tak aby czas wykonywania jakiejś operacji był dłuższy. Odbiorca monitorując czas reakcji systemu/procesu może stwierdzić, kiedy wiadomość została zasygnalizowana.

c) **Kanały terminacyjne** – polegające na uruchamianiu i wyłączaniu procesów w systemie w określonych odstępach czasowych. Jeżeli np. proces zakończył się o określonym czasie można uznać iż przesłano wartość 1, w przeciwnym razie 0.

d) **Kanały wyczerpywania zasobów** – wartość 1 lub 0 jest wyznaczana przez stan zajętości danego zasobu w systemie – np. zapełnienia dysku twardego, zajętości procesora, itp.

e) **Kanały mocy** – bity informacji mogą być nadawane poprzez cykliczne zmiany poboru prądu przez urządzenie.

Po dokładniejszej analizie wyżej wymienionych typów – każdy z nich można w szerszym ujęciu zakwalifikować jako kanał pamięciowy lub czasowy (np. terminacyjny czy mocy – gdyż wymagają one zmian w określonym przedziale czasu). Kemmerer [11] definiuje podstawowe kryteria, jakie muszą zaistnieć, aby możliwe było stworzenie kanału każdego z tych dwóch typów:

W przypadku kanału pamięciowego:

1. Proces nadawcy i odbiorcy musi mieć dostęp do tych samych współdzielonych zasobów.
2. Musi istnieć możliwość modyfikacji współdzielonych zasobów przez proces nadawcy.
3. Musi istnieć możliwość wykrycia przez proces odbiorcy zmiany w zasobach.
4. Musi istnieć mechanizm służący do inicjowania komunikacji pomiędzy nadawcą i odbiorcą tak, aby możliwe było prawidłowe synchronizowanie zdarzeń. Może być to inny kanał ukryty o mniejszej przepustowości.

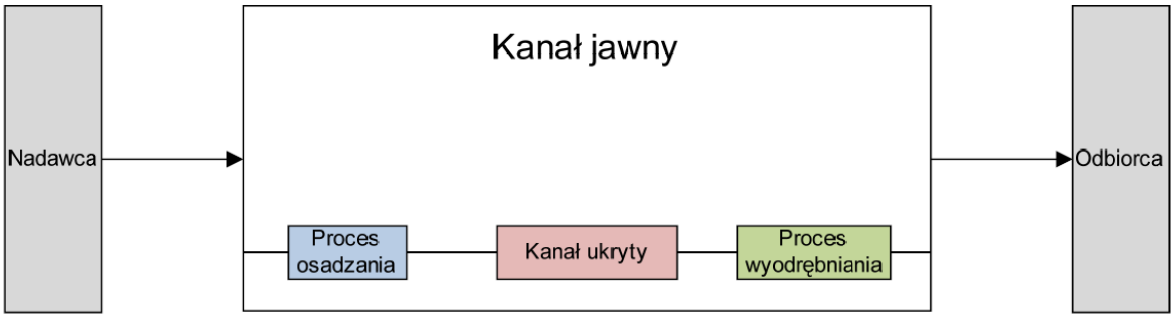
W przypadku kanału czasowego:

1. Proces nadawcy i odbiorcy musi mieć dostęp do tych samych współdzielonych zasobów.
2. Proces nadawcy i odbiorcy musi mieć dostęp do referencji czasu – np. zegara czasu rzeczywistego.

3. Nadawca musi mieć możliwość zmiany czasu reakcji systemu (w odniesieniu do współdzielonego zasobu) w taki sposób, aby była ona zauważalna przez odbiorcę.
4. Musi istnieć mechanizm pozwalający na zasygnalizowanie rozpoczęcia procesu przesyłania danych.

Przy użytkowaniu ukrytego, pamięciowego kanału informacyjnego obie strony komunikacji mogą jawnie ze sobą wymieniać dane na zasadach ściśle określonych w polityce bezpieczeństwa danego systemu (polityka ta może zabraniać np. wymiany pewnych rodzajów informacji, co stwarza pole do zastosowań kanałów ukrytych). Jednocześnie korzystają one z techniki zwanej steganografią (z greckiego: *ukryte pismo*), polegającej na modyfikacji określonych danych w taki sposób aby bez widocznej zmiany ich charakteru ukryć w nich "sekretne" informacje możliwe do odczytania jedynie dla wybranych osób znających klucz lub odpowiedni algorytm. Potocznie nazywa się to *szmuglowaniem* danych.

Przedstawiony na Rys. 2 koncepcyjny model ukrytego kanału informacyjnego jest bardzo podobny do modelu kanału "jawnego" opisanego wcześniej. Z tą różnicą, iż miejsce nadajnika/odbiornika zajmują tutaj moduły odpowiedzialne za steganograficzne osadzenie oraz wyodrębnienie informacji z jawnego strumienia danych przesyłanych pomiędzy nadawcą oraz odbiorcą.



Rys. 2 Model koncepcyjny ukrytego kanału informacyjnego

Bardzo podobne również są cechy, które charakteryzują kanały ukryte, a są to:

a) **Pojemność** – całkowita ilość informacji, które można przesłać poprzez kanał
b) **Przepustowość** – ilość informacji, jaką można przesłać przez kanał w danej jednostce czasu, przy czym powszechnie przyjmuje się, iż im mniejsza przepustowość takiego kanału tym mniejsze szanse na jego wykrycie (konieczność wprowadzania mniejszej ilości zmian do kanału „jawnego", które mogą zostać zauważone).
c) **Zakłócenia** – liczba zaburzeń, które mogą ingerować w informacje przesyłane kanałem.
d) **Tryb transmisji** – synchroniczny lub asynchroniczny.

1.3 Sieciowe ukryte kanały informacyjne – specyfikacja problemu

W obliczu coraz większej popularności Internetu bardzo dużą wagę zaczyna się przywiązywać do bezpieczeństwa przesyłanych informacji oraz prywatności użytkowników korzystających z sieci. Hintz [9] określił dwie grupy zagrożeń dla ujawnienia danych przesyłanych przez sieci komputerowe, mianowicie:

- Zagrożenie przypadkowego obserwatora – osoby, która wybiórczo monitoruje ruch sieciowy w danym segmencie, wyszukując jedynie określone wzorce (np. słowa kluczowe) i dysponująca zbyt małą mocą obliczeniową do bardzo szczegółowego monitorowania każdego z połączeń z osobna.
- Zagrożenie obserwatora specjalizowanego – który skupia się na szczegółowym monitorowaniu kilku podejrzanych osób. Dysponuje on środkami pozwalającymi na pełne przechwytywanie i analizę każdego połączenia sieciowego.

Najpowszechniejszym i najpewniejszym rozwiązaniem chroniącym przed każdym z tych zagrożeń jest kryptografia - czy to na poziomie wymienianych przez użytkowników danych czy już samych protokołów sieciowych. Czasem jednak jej użycie, zwłaszcza gdy ruch sieciowy jest monitorowany przez obserwatora specjalizowanego może nie być najlepszym rozwiązaniem, gdyż:

- Używanie kryptografii może być w danym miejscu nielegalne (np. niektóre kraje totalitarne).
- Algorytmy kryptograficzne mogą mieć tylne furtki umożliwiające ich złamanie.
- Używanie kryptografii może wzbudzać podejrzenia.

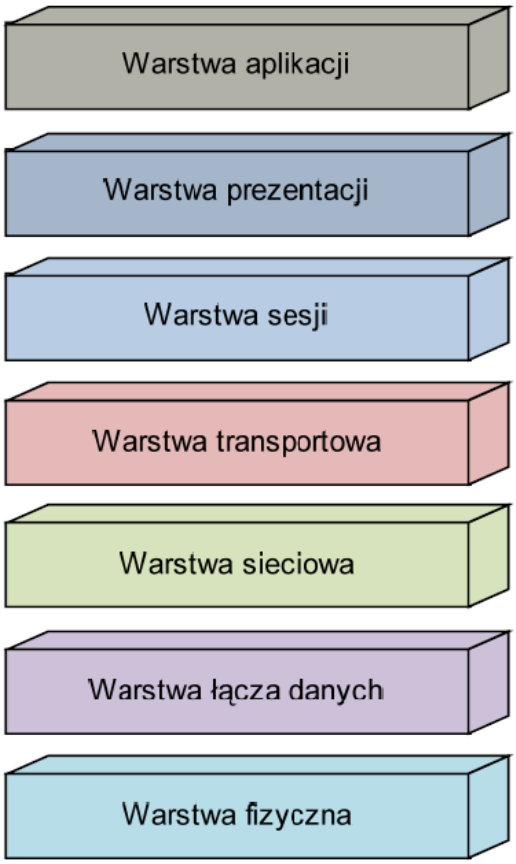
Użycie ukrytych kanałów informacyjnych zapewnia w określonym stopniu bezpieczeństwo oraz tajność samego faktu przesyłania danych poprzez fragmenty sieci narażone na podsłuch oraz manipulację danych. Wspomniana wcześniej cecha każdego takiego kanału – mianowicie naruszanie polityki bezpieczeństwa danego systemu może sprawić, iż technologia ta może zostać wykorzystana w sieciach, których twórcy za wszelką cenę chcą kontrolować przepływ informacji. Dotyczy to zarówno aspektów negatywnych – szpiegostwa – jak i (względnie) pozytywnych – ochrony prywatności lub walki z cenzurą Internetu w krajach łamiących prawa człowieka. Z punktu widzenia komunikacji sieciowej – omówione w punkcie poprzednim ukryte kanały informacyjne mogą z powodzeniem wykorzystywać jako nośnik danych pakiety sieciowe, wymieniane pomiędzy różnymi węzłami sieci w czasie prowadzenia nie wzbudzających podejrzeń transmisji (pierwsza publikacja na ten temat pojawiła się dopiero w roku 1987).

2 Protokoły sieciowe - IP i TCP

Aby możliwa była analiza istniejących rozwiązań technicznych dot. sieciowych kanałów informacyjnych niezbędne jest omówienie ogólnego modelu sieciowego OSI oraz podstawowych szczegółów dotyczących architektury i algorytmów stosowanych w protokołach TCP/IP.

2.1 Model sieciowy OSI

Sieciowy model OSI (*ang. Open System Interconnection*) [18] jest standardem zdefiniowanym przez organizację ISO, opisującym strukturę komunikacji sieciowej. Traktowany jest on jako wzorzec do tworzenia wszelkich rodzin protokołów komunikacyjnych i współpracy pomiędzy nimi. Jego podstawowym założeniem jest podział każdego systemu sieciowego na siedem, współpracujących ze sobą warstw, przedstawionych na Rys. 3.



Rys. 3 Model sieciowy OSI

Każda z warstw OSI jest warstwą abstrakcji, definiującą konkretne zadania oraz rodzaje danych jakie mogą być wymieniane z warstwami sąsiednimi. Trzy górne warstwy często przedstawiane są w postaci jednej – Warstwy Aplikacji, na poziomie której działają aplikacje widoczne i użyteczne dla człowieka - klienci sieciowi, serwery (np. WWW, FTP), itp. Najbardziej interesującymi z naszego punktu widzenia warstwami są warstwy transportowa oraz sieciowa.

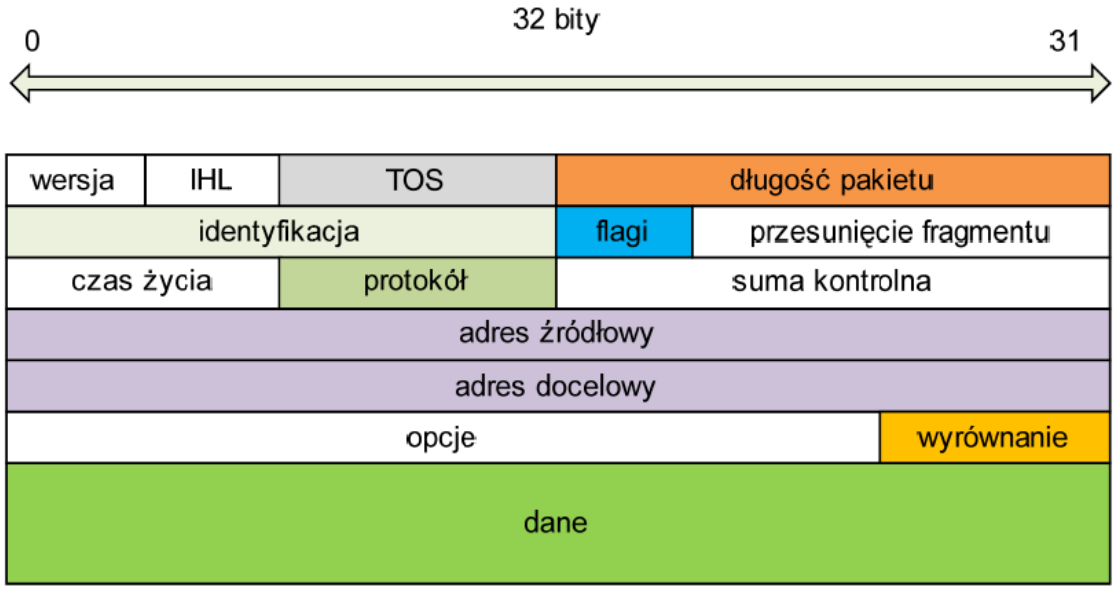
Warstwa sieciowa lub inaczej warstwa protokołu internetowego (*IP – Internet Protocol*) jest fundamentem działania Internetu. Na jej poziomie każdy użytkownik (komputer, inne urządzenie komunikacyjne) sieci ma przypisany unikalny adres dzięki któremu możliwe jest wyznaczenie drogi do niego (routing) i przesyłanie danych. Na tym poziomie prowadzona jest jedynie komunikacja pomiędzy urządzeniami, bez rozróżniania poszczególnych sesji. Również mechanizmy zapewnienia integralności przesyłanych danych ograniczone są jedynie do detekcji błędów, bez możliwości ich korekcji [18].

Zadaniem **warstwy transportowej** jest niezawodne dostarczanie danych (ze szczególnym zwróceniem uwagi na kolejność ich napływania) oraz ich kierowanie do właściwych sesji na uczestnikach połączenia, dzięki wykorzystaniu mechanizmu portów. Mechanizm ten zabezpiecza różne transmisję danych, prowadzonych między tymi samymi urządzeniami, przed interferencją między sobą. Także na poziomie tej warstwy funkcjonują mechanizmy nawiązywania oraz zamykania połączeń pomiędzy uczestnikami transmisji danych, jak również detekcji oraz co najważniejsze korekcji błędów.

Mimo, iż teoretycznie możliwe jest stworzenie ukrytego kanału informacyjnego w dowolnej z wyżej wymienionych warstw modelu OSI, niniejsza praca skupiać się będzie na dwóch, powyżej wymienionych warstwach modelu. Powodowane jest to faktem, iż o ile w ramach sieci, jaką jest Internet panuje pełna różnorodność doboru standardów sprzętu sieciowego (warstwa fizyczna OSI) czy aplikacji, o tyle wspólnym elementem niemal wszystkich urządzeń mających dostęp do globalnej sieci jest konieczność obsługi pełnego standardu protokołów TCP/IP. Każdy komputer podpięty do Internetu korzysta z jednakowych warstw transportowych i sieciowych, dzięki czemu zagadnienie ukrytych kanałów informacyjnych dotyczy niemal każdego sieciowego systemu informatycznego na świecie.

2.2 Protokół IP

Protokół IP został ustandaryzowany w roku 1984 w dokumencie RFC-791 [5] specyfikującym wszystkie jego struktury danych oraz mechanizmy niezbędne do prowadzenia komunikacji. Podstawową jednostką transmisji danych w IP (zresztą jak w niemal każdym innym protokole sieciowym) jest pakiet - składający się z nagłówka oraz następującej po nim zawartości (danych). Budowa pojedynczego pakietu IP przedstawiona jest na Rys. 4. Poniżej zaś znajduje się omówienie poszczególnych pól nagłówka IP. Szczególna uwaga poświęcona została polom dodatkowym i rzadko używanym, jako że mogą one zostać z dużym prawdopodobieństwem wykorzystane w późniejszej pracy przy projektowaniu algorytmów ukrytego przesyłania danych.

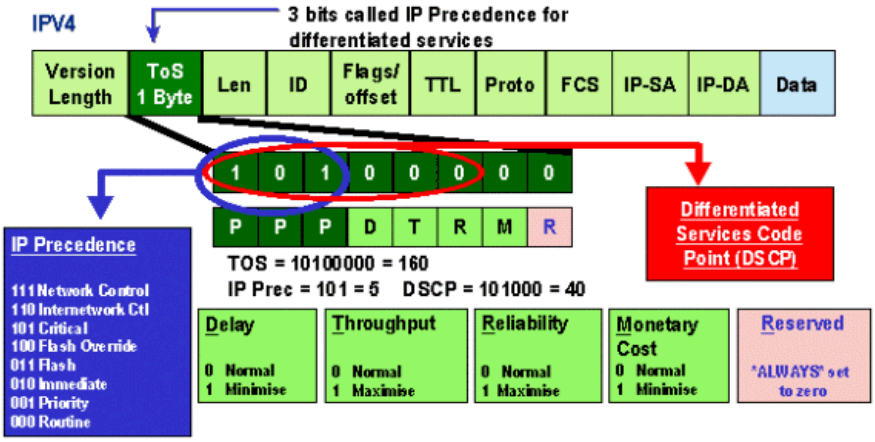


Rys. 4 Budowa pakietu protokołu IP

W skład nagłówka IP wchodzą następujące pola:

1. **Wersja** – 4 bitowe pole zawierające numer wersji protokołu IP. Najpowszechniej używaną wersją jest wersja 4, która w długiej perspektywie czasu zostanie wyparta przez protokół w wersji 6. Wszystkie omawiane w tej pracy pola i mechanizmy protokołu IP dotyczyć będą jego wersji czwartej.
2. **IHL** (*ang. Internet Header Length*) – 4 bitowe pole zawierające całkowitą długość nagłówka IP liczoną w 32 bitowych słowach. Minimalną dopuszczalną wartością tego pola jest 5 oznaczającą nagłówek o długości 5*32=160 bitów = 20 bajtów. Z tego wynika również, iż maksymalną wartością tego pola może być 15 oznaczające maksymalną dopuszczalną długość nagłówka wynoszącą 15*32=480 bitów = 60 bajtów.
3. **TOS** (*ang. Type Of Service*) – 8 bitowe pole wskazujące abstrakcyjny zestaw parametrów łącza, wymagany do przeprowadzenia transmisji. Parametry te ustawiane są przez urządzenia sieciowe przez które transmitowany jest dany pakiet. Dzięki odpowiedniej zawartości tego pola niektóre transmisje mogą być np. traktowane priorytetowo. Zasadniczy wybór pomiędzy parametrami łącza polega na wskazaniu czy nadawcy bardziej zależy na zminimalizowaniu opóźnień w transmisji, zmaksymalizowaniu szybkości transmisji czy zapewnieniu jak największej niezawodności.

   Zawartość pola TOS rozkłada się tak, jak przedstawiono na Rys. 5.



Rys. 5 Zawartość pola TOS pakietu IP, źródło: http://www.cisco.com/warp/public/473/152.html

4. **Długość pakietu** – 16 bitowe pole zawierające całkowitą długość pakietu (nagłówek + dane) liczoną w oktetach (8 bitów). Maksymalną wartością tego pola jest 2^16 = 65535 oktetów – tyle wynosi techniczna bariera wielkości pakietu IP.
5. **Identyfikacja** – 16 bitowa wartość przypisywana przez nadawcę, pomocna przy składaniu pofragmentowanych pakietów.
6. **Flagi** – 3 bity. Flagi kontrolne używane przez mechanizm fragmentacji pakietów IP. Zawartość tego pola rozkłada się następująco:

   Bit 0 – zarezerwowany, musi zawsze być zerem

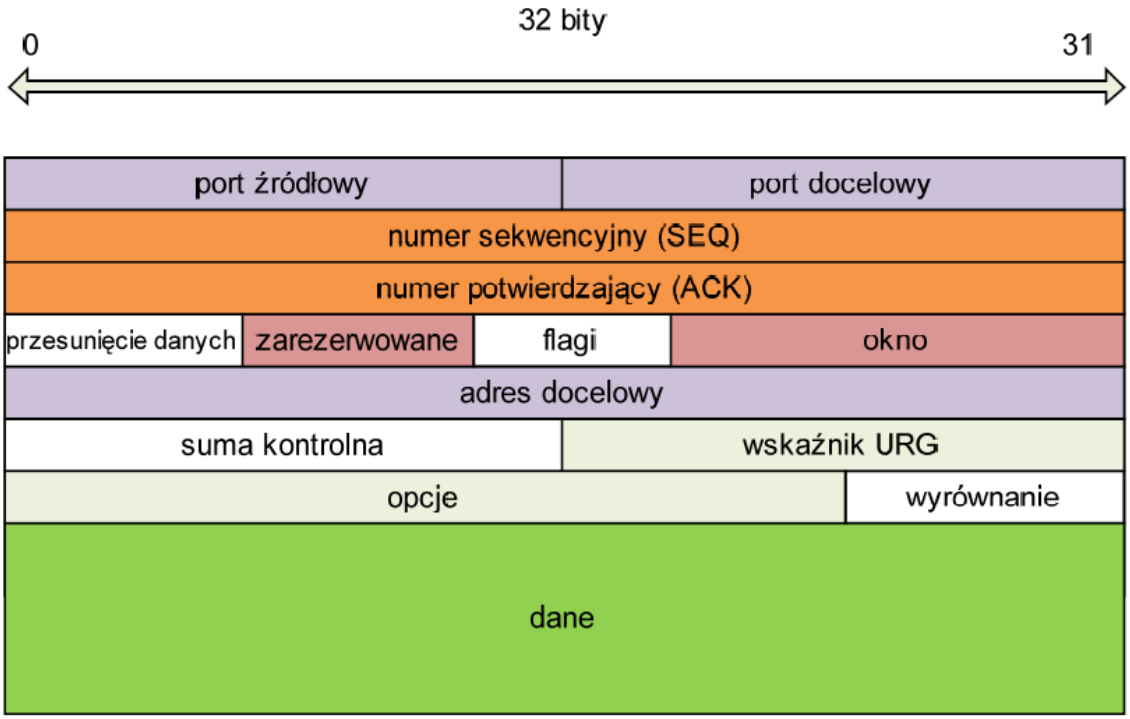
   Bit 1 (DF) – 0 – pakiet może zostać zfragmentowany, 1 – nie może

   Bit 2 (MF) – 0 – ostatni fragment, 1 – więcej fragmentów w drodze
7. **Przesunięcie fragmentu** – pole 13 bitowe określające do którego miejsca w pakiecie przynależy dany fragment. Przesunięcie fragmentu wyrażane jest w jednostkach 8-oktetowych (64 bity).
8. **Czas życia (TTL)** – 8 bitowe pole określające przez jaką maksymalną liczbę routerów może przejść dany pakiet. Każdy router, przez który przechodzi dany pakiet IP zmniejsza wartość tego pola o 1. Gdy jego wartość osiągnie 0, pakiet jest niszczony- ma to zapobiec błądzeniu po sieci pakietów, które z różnych przyczyn nie mogą zostać doręczone lub wpadają w pętlę.
9. **Protokół** – 8 bitowe pole określające, który protokół wyższej warstwy transportowany jest przez dany pakiet IP. Numery protokołów są ustandaryzowane i znajdują się w dokumencie RFC1700 [7].
10. **Suma kontrolna nagłówka** – 16 bitowe pole. Suma kontrolna, używana do kontroli integralności, będąca jedynkowym dopełnieniem jedynkowodopełnieniowej sumy wszystkich 16 bitowych słów nagłówka. Przyjmuje się, iż przy wyznaczaniu sumy kontrolnej wartość tego pola wynosi 0.
11. **Adres źródłowy** – 32 bity. Adres sieciowy nadawcy pakietu.
12. **Adres docelowy** – 32 bity. Adres sieciowy odbiorcy pakietu.

13. **Opcje** dodatkowe – pole o zmiennej długości (nie jest ono obligatoryjne), zawierające dodatkowe opcje IP, wykorzystywane przez niektóre urządzenia sieciowe. Sposób kodowania tych opcji w nagłówku oraz ich dokładna charakterystyka znajdują się w rozdziale 3.1 dokumentu RFC0791 [5].
14. **Wyrównanie** - część nagłówka wypełniona zerami, używana w niektórych przypadkach tak, aby rozmiar nagłówka był wielokrotnością 32 bitów.
15. **Dane** - dane przenoszone przez pakiet – pierwszym bajtem danych jest pierwszy bajt nagłówka protokołu warstwy wyższej (w naszym przypadku TCP).

2.3 Protokół TCP

Protokół TCP (Transmission Control Protocol) został ustandaryzowany w roku 1981 w dokumencie RFC793 [6]. Obecnie niemal zawsze jest on używany do komunikacji sieciowej wraz z protokołem IP, dlatego bardzo często używa się określenia *protokół TCP/IP*. Budowa pojedynczego pakietu TCP przedstawiona jest na Rys. 6. Poniżej zaś znajduje się omówienie poszczególnych pól.



Rys. 6 Budowa pakietu protokołu TCP

W skład nagłówka TCP wchodzą następujące pola:

1. **Port źródłowy** – 16 bitowe pole określające port źródłowy połączenia TCP.
2. **Port docelowy** – 16 bitowe pole określające port docelowy połączenia TCP.
3. **Numer sekwencyjny (SEQ)** – 32 bitowe pole określające numer sekwencyjny pierwszego bajtu danych przesyłanych w pakiecie. Używane jest również przy nawiązywaniu nowego połączenia, co zostanie omówione później.
4. **Numer potwierdzający (ACK)** – 32 bitowe pole określające numer sekwencyjny, którym nadawca pakietu potwierdza drugiej stronie transmisji odebranie poprzednich danych lub pakietu.
5. **Przesunięcie danych** – 4 bitowe pole. Przesunięcie względem początku pakietu TCP wyrażone w słowach 32 bitowych wskazujące na miejsce, gdzie rozpoczynają się przesyłanie w pakiecie dane.
6. **Zarezerwowane** – 6 bitowe pole, które zawsze musi być wypełnione zerami.
7. **Flagi** – 6 bitowe pole zawierające flagi – URG, ACK, PSH, RST, SYN, FIN używane do nawiązywania, kończenia i zarządzania połączeniem. Ich działanie jest omówione w dalszej części dokumentu.
8. **Okno** – 16 bitowe pole, w którym nadawca pakietu informuje odbiorcę o maksymalnym rozmiarze danych, które może on odebrać w danej chwili. Zazwyczaj podyktowane jest ono ograniczeniami pamięciowymi danej maszyny - chroni to przed nadmiernym obciążeniem którejkolwiek ze stron transmisji.
9. **Suma kontrolna** – 16 bitowe pole zawierające sumę kontrolną pakietu. Wyznaczana jest ona poprzez jedynkowe dopełnienie jedynkowodopełnieniowej sumy wszystkich 16 bitowych słów nagłówka i zawartości pakietu. Jeżeli rozmiar nagłówka i zawartości nie jest wielokrotnością 16 bitów dane uzupełnia się na końcu zerami. Przy wyznaczaniu sumy kontrolnej przed nagłówkiem TCP umieszcza się tzw. pseudonagłówek zawierający pola przepisane z nagłówka IP – adres źródłowy, docelowy, numer protokołu oraz rozmiar nagłówka i danych TCP. Zabezpiecza to przed przetworzeniem pakietu, który dana maszyna odebrała przypadkowo, a który nie był przeznaczony dla niej.
10. **Wskaźnik URG** – 16 bitowe pole zawierające wskaźnik do danych przesyłanych w trybie pilnym. Pole to jest wykorzystywane jedynie wtedy, gdy aktywna jest flaga URG.
11. **Opcje dodatkowe** – pole o zmiennej długości (nie jest ono obligatoryjne), zawierające dodatkowe opcje TCP, wykorzystywane do sterowania transmisją. Sposób kodowania tych opcji w nagłówku oraz ich dokładna charakterystyka znajdują się w rozdziale 3.1 dokumentu RFC0793 [6].
12. **Wyrównanie** - część nagłówka wypełniona zerami, używana w niektórych przypadkach tak aby rozmiar nagłówka był wielokrotnością 32 bitów.

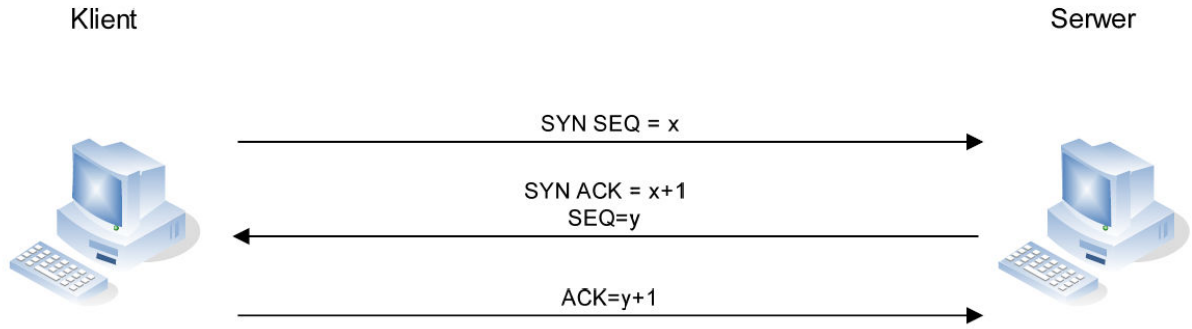
2.3.1 Mechanizmy protokołu TCP

2.3.1.1 Nawiązywanie połączenia

Nawiązanie nowego połączenia pomiędzy dwoma stronami transmisji rozpoczyna się w momencie, gdy strona aktywna (klient) wysyła do strony pasywnej (serwera) pakiet TCP bez zawartości, za to z włączoną flagą SYN, oraz losowym numerem sekwencyjnym w polu SEQ – nazywanym numerem ISN. Serwer musi potwierdzić przyjęcie pakietu SYN od klienta i wysłać własny pakiet SYN, zawierający początkowy numer sekwencyjny (w polu SEQ) danych, które serwer będzie wysyłać przez to połączenie. W tym samym pakiecie, w polu ACK serwer wysyła również potwierdzenie numeru sekwencyjnego klienta. W ostatniej fazie nawiązywania połączenia klient potwierdza przyjęcie pakietu SYN od serwera wysyłając pakiet ACK.

"Numer wysyłanego potwierdzenia ACK jest następnym numerem kolejnym oczekiwanym przez nadawcę potwierdzenia ACK. Ponieważ segment SYN jest numerowany tak jak pojedynczy bajt, więc numerem potwierdzenia w komunikacie ACK jest początkowy numer kolejny zwiększony o jeden" [19]

Jako, że do nawiązania połączenia potrzebna jest wymiana 3 pakietów. Algorytm ten nazywany jest "uzgadnianiem trójfazowym". Przedstawione jest to na Rys. 7.

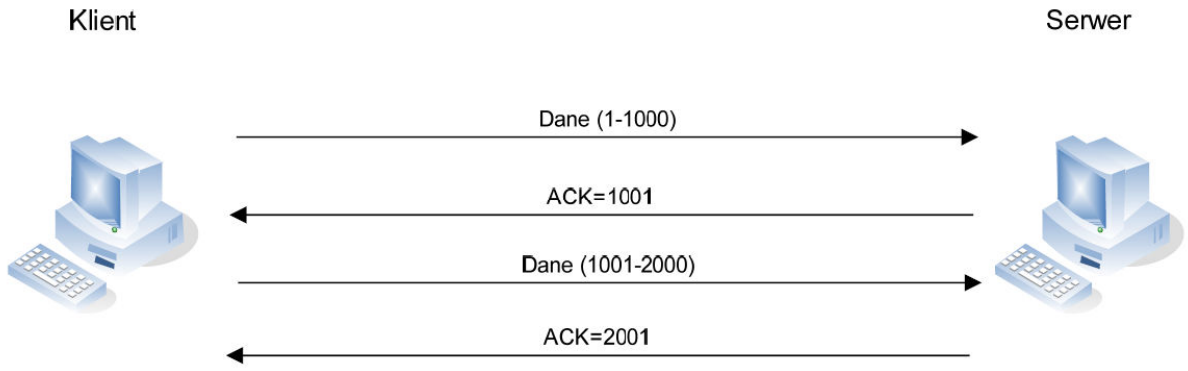


Rys. 7 Trójfazowe nawiązanie połączenia TCP

2.3.1.2 Prowadzenie komunikacji

Po nawiązaniu komunikacji następuje wymiana danych pomiędzy obiema stronami transmisji. W celu zapewnienia pełnej kontroli nad przesyłanymi danymi i tym samym zapewnieniu mechanizmu detekcji i korekcji błędów, przy wymianie danych używa się numerów sekwencyjnych w polach SEQ i ACK. Zapewniają one skuteczne zabezpieczenie przed pakietami zagubionymi lub takimi, które są nadsyłane w nieprawidłowej kolejności.

Fundamentalną regułą TCP/IP jest to, iż każdy przesyłany bajt danych posiada swój numer sekwencyjny i tym samym jego odebranie może zostać przez drugą stronę komunikacji potwierdzone. Każdy pakiet z danymi i numerem SEQ zostaje potwierdzony przez odbiorcę pakietem ACK zawierającym numer SEQ odebranego pakietu powiększony o liczbę odebranych bajtów +1. Następny pakiet, który przybędzie do odbiorcy będzie miał numer SEQ taki sam, jak potwierdzenie ACK. Przykład ten ilustruje Rys. 8.



Rys. 8 Wymiana danych w TCP

Numery sekwencyjne dla każdej ze stron połączenia są niezależne. W momencie, gdy dla wysłanego pakietu nie otrzymano potwierdzenia jego odebrania, uznaje się, iż został on utracony i wysłany zostaje ponownie. Ilość ponowień transmisji po których uznaje się, iż połączenie zostało utracone zależy ściśle od systemu operacyjnego i implementacji protokołu TCP.

2.3.1.3 Zamykanie połączenia

Zamykanie aktywnego połączenia od jego nawiązania różni się niewiele. Strona, która jako pierwsza chce zamknąć połączenie wysyła pusty pakiet z ustawioną flagą FIN i numer SEQ właściwymi dla połączenia. Druga strona połączenia odsyła pakiet FIN z polem ACK potwierdzającym odebranie pakietu FIN. Od tego momentu strona, która zamknęła połączenie nie może nic wysłać, ale nadal może odbierać dane od drugiej strony połączenia. Dopiero, gdy ta wyśle pakiet FIN i otrzyma potwierdzenie jego odebrania – połączenie TCP uznaje się za zamknięte całkowicie.

2.3.1.4 Resetowanie połączeń

Opcja resetowania połączenia używana jest w przypadku wystąpienia w komunikacji anomalii, która nie może zostać naprawiona. W przypadku otrzymania pakietu uznanego

za anomalię maszyna odsyła w odpowiedzi pakiet z ustawioną flagą RST i numerem ACK będącym odpowiedzią na numer SEQ. Po wystąpieniu takiego zdarzenia obie strony komunikacji uznają połączenie za awaryjnie zamknięte.

Sytuacje na które maszyna może odpowiedzieć pakietem RST:

- Otrzymano pakiet skierowany do nieobsługiwanego portu.
- Nadesłany pakiet nie pasuje do aktualnego stanu połączenia (np. przy próbie wysłania czegokolwiek przed trójstopniowym nawiązaniem połączenia).

3 Istniejące rozwiązania w zakresie ukrytych kanałów w TCP/IP

Niemal wszystkie prace poświęcone ukrytym kanałom informacyjnym w sieciach TCP/IP skupiają się na kanałach pamięciowych. Spowodowane jest to faktem, iż w nagłówkach IP i TCP znajdują się pola lub fragmenty zarezerwowane, nieużywane lub takie, w których możliwe jest ukrycie własnych danych bez zakłócania transmisji. Dotyczy to zarówno sztucznego ruchu sieciowego generowanego jedynie po to, aby ukryć w nim jakieś informacje, jak i wplatania danych w pakiety należące do legalnego ruchu, już istniejącego w sieci.

Rowland [15] zaproponował możliwość ukrywania danych w polu ID nagłówka IP, które normalnie zawiera unikalny numer pakietu używany przy jego ewentualnej fragmentacji. W zaprezentowanym przez niego przykładzie, w 16 bitowym polu ID umieszczane były zakodowane kolejne wartości ASCII liter wchodzących w skład przesyłanej wiadomości. W identyczny sposób przeniósł on ten pomysł na 32 bitowe numery ISN używane przy nawiązywaniu połączenia TCP. Jako, że tak trywialna metoda jest zbyt łatwa do wykrycia (pole ID w zależności od implementacji protokołu IP albo jest losowe albo sekwencyjnie rośnie przy każdym nowym pakiecie), Ahsan [1] zaznaczył możliwość ukrywania danych (dodatkowo zaszyfrowanych) w starszym bajcie pola ID, przy jednoczesnym losowaniu zawartości bajtu młodszego.

Ahsan zwrócił również uwagę na możliwości wykorzystania flagi DF (*Don't Fragment*) w nagłówku IP, zezwalającej routerom na fragmentowanie pakietów – manipulowanie jej wartością pozwalałoby na przesłanie 1 bita danych na pakiet. Jednak obecne implementacje protokołu TCP/IP ściśle definiują kiedy flaga DF powinna być włączona a kiedy nie, a wykrycie jakichkolwiek odstępstw może z dużym prawdopodobieństwem wskazywać na celowe działania.

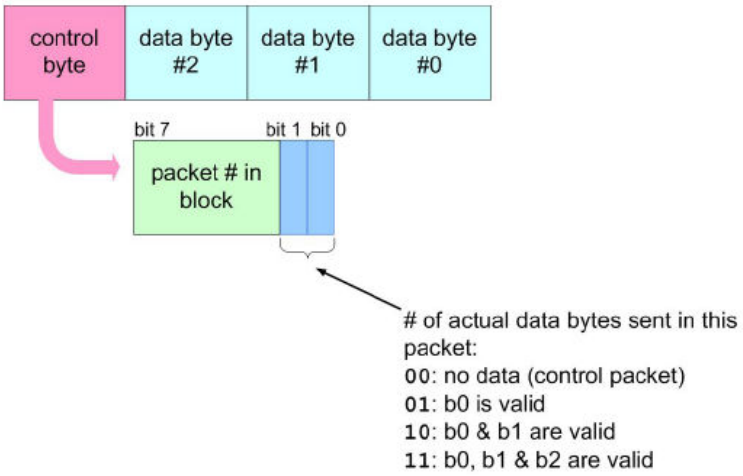
Z równie łatwych do wykrycia, acz pozwalających na transmitowanie znacznie większych ilości informacji metod, Ahsan zwrócił również uwagę na pola ACK oraz URG w pakietach TCP. W momencie, gdy flagi protokołu, odpowiednio również ACK i URG, są wyłączone odpowiadające im pola (32 bitowe) nie są używane i domyślnie przyjmują wartość 0. W takich przypadkach w polach tych, bez zakłócania działania protokołu TCP można umieścić i przetransportować dowolne dane.

Qu, Su i Feng [20] zaproponowali natomiast podmienianie danych w polu TTL pakietu IP na wcześniej wyznaczone wartości (z uwzględnieniem zakłóceń, jakie wprowadza do TTL zmienna trasa pakietu), zapewniające przesłanie w pojedynczym pakiecie jednego bitu ukrytej informacji. Pomysł ten został głębiej przeanalizowany przez Zandera, Armitage'a i Brancha [21], którzy przebadali statystyczny rozkład wartości TTL w pakietach IP docierających do różnych miejsc w Internecie i zauważyli, iż przeciętna jej wariacja jest bardzo nieduża

i zazwyczaj ogranicza się do 2 różnych wartości w niedużym odstępie od siebie. Tym samym, z punktu widzenia osoby monitorującej sieć, znaczące różnice w wartości TTL pakietów pochodzących z jednej maszyny mogą wskazywać na istniejący ukryty kanał.

Inne natomiast rozwiązanie opracował zespół Johna Giffina [8] badając możliwości ukrywania danych w polach TCP Timestamp. Opcja Timestamp w protokole TCP pozwala na przesyłanie w pakiecie SYN 32 bitowego znacznika czasu, który następnie wraca do nadawcy w pakiecie SYN/ACK. Pozwala to na oszacowanie czasu jaki pakiet potrzebuje na przebycie drogi tam i z powrotem. Ideą wykorzystania tego mechanizmu jest modyfikacja najmłodszego bitu znacznika – dzięki temu w każdym pakiecie zawierającym opcję Timestamp można przesłać jeden bit ukrytej informacji. Mocną stroną tego rozwiązania jest fakt, iż ostatni bit znacznika czasu reprezentuje wartości rzędu milisekund – praktycznie losowe we współczesnych systemach. Dzięki temu odpowiednie kodowanie ukrytych danych na tym bicie nie wprowadza obserwowalnych anomalii i czyni tą metodę skrajnie trudną do wykrycia.

Z metod pamięciowych najbardziej, jak dotąd, zawansowaną techniką jest NUSHU – opracowana przez Joannę Rutkowską [16]. NUSHU opiera się na ukrywaniu danych w 32 bitowych numerach ISN wymienianych między maszynami przy inicjalizacji połączenia TCP (pakiety z flagami SYN i SYN/ACK). Jednak w odróżnieniu od metody zaproponowanej przez Rowlanda [15], NUSHU nie generuje sztucznego ruchu sieciowego oraz za pomocą wbudowanego, własnego protokołu zapewnia kontrolę transmisji i korekcję błędów, które mogą wystąpić.



Rys. 9 Struktura pakietu NUSHU, źródło: [16]

Na Rys. 9, zaczerpniętym z pracy Joanny Rutkowskiej [16] pokazana jest struktura pojedynczego, 32 bitowego pakietu osadzanego w miejsce numeru ISN. Pierwsze 8 bitów zajmuje pole służące do kontroli działania protokołu, 24 pozostałe bity służą do przenoszenia danych. Aby zapewnić kontrolę nad przesyłanymi pakietami, każdy przesyłany bajt posiada swój 7 bitowy numer sekwencyjny, jednak z powodu dużych ograniczeń tego pola, dane muszą być grupowane w osobne bloki. Każdy nowy blok poprzedzony zostaje pakietem kontrolnym ISN_NEW_BLOCK, który resetuje numer sekwencyjny i rozpoczyna zliczanie oraz potwierdzanie pakietów od nowa. ISN_NEW_BLOCK zostaje wysłany tylko wtedy, gdy zostało potwierdzone odebranie wszystkich pakietów z poprzedniego bloku.

NUSHU jest protokołem pasywnym, to znaczy iż strona odbierająca dane nie musi odsyłać z powrotem żadnych informacji. Korzystając z własności protokołu TCP nadawca wie, iż pakiet został dostarczony poprawnie jeżeli otrzyma od stosu TCP/IP odbiorcy pakiet potwierdzający nawiązanie połączenia numerem ISN (pakiet SYN/ACK). Jeżeli z któryś z pakietów zostanie zgubiony, będzie on retransmitowany tak długo aż odbiorca go nie otrzyma i potwierdzi.

Jako, iż numery ISN są używane przez protokół TCP do kontroli przebiegu połączenia, jakakolwiek ich modyfikacja spowoduje jego rozsynchronizowanie i w konsekwencji zerwanie. Aby tego uniknąć, NUSHU utrzymuje własne tablice połączeń TCP i „prawdziwych" numerów ISN oraz aktualnych numerów sekwencyjnych pakietów. Przed przekazaniem pakietu do stosu TCP numery sekwencyjne są modyfikowane przez moduł NUSHU tak, aby odzwierciedlały prawdziwy stan połączenia.

Pseudolosowość numerów ISN sprawia, iż jakiekolwiek pojawiające się w nich wzorce i regularności mogą zostać łatwo wychwycone i wskazywać na istnienie ukrytego kanału informacyjnego. Wbudowany w NUSHU mechanizm kryptografii stara się wprowadzić jak największą entropię do kanału - algorytm DES służy do wygenerowania klucza w oparciu o parametry połączenia TCP (adresy i porty nadawcy i odbiorcy), który następnie jest XOR-owany z zawartością pakietu. Zmienność numerów portów w przypadku nawiązywania każdego nowego połączenia zapewnia iż każdy pakiet NUSHU zostanie zaszyfrowany innym kluczem.

Mimo zastosowania kryptografii, Badania Tumoian'a i Anikeev'a [22] pokazują jednak, iż detekcja ukrytych kanałów informacyjnych opartych o NUSHU jest wykonalna za pomocą analizy rozkładu statystycznego numerów ISN za pomocą sieci neuronowych.

Zupełnie inaczej ma się sprawa czasowych ukrytych kanałów informacyjnych w TCP/IP. Ze względu na spore techniczne trudności, istnieje bardzo niewiele badań i publikacji na ten temat. Istota istnienia tego typu kanałów opiera się na monitorowaniu opóźnień w dostarczaniu pakietów od nadawcy do odbiorcy. Dzięki możliwości wpływania na odstępy czasowe w jakich pakiety są doręczane, możliwe jest przekazywanie odbiorcy wiadomości poszczególnych wartości 1 lub 0. Prawdopodobnie jedyną szerszą publikacją omawiającą to zagadnienie jest praca Cabuka i in. [3]. Niestety ze względu na możliwe czynniki wpływające na opóźnienia pakietów (wydajność infrastruktury, obciążenia sieci, różne trasy routingu), problemy z synchronizacją przez obie strony komunikacji oraz wysoce prawdopodobne dławienie „legalnego" ruchu sieciowego zastosowanie tego typu kanałów jest mało realne w praktyce.

Poza wyżej wymienionymi rozwiązaniami, na przestrzeni lat zostało zaproponowanych jeszcze kilka metod ukrywania danych w protokole TCP/IP, które można podzielić na dwie grupy. Pierwszą z nich jest korzystanie z jednobitowych flag i pól, które są nieużywane lub nie powodują większych zmian w działaniu protokołu – np. korzystanie z flag TOS w nagłówkach IP, zaznaczone przez Murdocha i Lewisa [13]. Niestety obecnie flagi te są albo domyślnie wyzerowane albo mają swoją stała, przedefiniowaną wartość w ramach danego systemu komputerowego i jakakolwiek ich zmiana może zostać łatwo wykryta.

Drugą grupą rozwiązań, która doczekała się wielu implementacji jest ukrywanie danych w pakietach ICMP i IGMP, będących protokołami pomocniczymi dla TCP/IP. Duża ilość typów pakietów oraz formatów pól tych protokołów sprawia, iż ukrywanie w nich danych jest trywialnie proste. Niestety specyfika ICMP jako protokołu służącego do diagnostyki sieci sprawia, iż ruch z jego użyciem jest znikomy, dlatego też niemal wszystkie implementacje ukrytych kanałów bazują na generowaniu ruchu sztucznego, który jest bardzo łatwy do wykrycia, zwłaszcza w przypadku stosowania rzadko stosowanych typów pakietów. Podobnie

rzecz tyczy się IGMP, który stosowany jest jedynie w specjalizowanych rodzajach systemów sieciowych. Kolejnym czynnikiem, który działa na niekorzyść tego typu rozwiązań jest fakt, iż zarówno ICMP jak i IGMP mogą (i zazwyczaj są) być filtrowane przez administratorów sieci komputerowych bez szkody dla normalnego ruchu sieciowego. Również przekazywanie pakietów przez routery stosujące maskowanie NAT może być bardzo utrudnione i uniemożliwiać zestawienie ukrytego kanału. Przykładami aplikacji realizujących ukryte połączenia poprzez pakiety ICMP są Ping Tunnel [14] oraz ICMP Tunnel [10]

Wszystkie istniejące rozwiązania i prace omawiające ukryte kanały informacyjne w TCP/IP – wymienione powyżej - posiadają kilka wspólnych, negatywnych cech:

- Z wyjątkiem NUSHU żadne nie zapewnia detekcji i korekcji błędów, co praktycznie wyklucza ich użycie do jakichkolwiek zastosowań z wyjątkiem prostej demonstracji zagadnienia. Żadne też (nawet NUSHU) nie zapewnia sesyjności transmisji.
- Wszystkie rozwiązania oferują jedynie transmisję jednokierunkową.
- Wiele przedstawionych technik bazuje na generowaniu sztucznego ruchu sieciowego, co sprawia iż prawdopodobieństwo zauważenia anomalii przez osoby próbujące wykryć istnienie ukrytego kanału jest bardzo wysokie.
- W żadnym przypadku nie zostały przeprowadzone badania dotyczące możliwej do uzyskania przepustowości danego kanału w warunkach rzeczywistych (tj. normalnego i nie wzbudzającego podejrzeń ruchu sieciowego).
- Każde z rozwiązań skupia się na wykorzystaniu tylko jednej z technik naraz. Brak badań dotyczących np. jednoczesnego wykorzystania metod ISN i IP ID.
- Brak szczegółowych badań dotyczących wykorzystania pól innych niż te wspomniane dotąd: flagi PSH w TCP, sztucznego sterowania rozmiarem pakietu, innymi opcjami TCP niż Timestamp, itp.
- Skupianie się jedynie na kryptografii jako sposobie na zmniejszenie prawdopodobieństwa wykrycia ukrytego kanału. Brak omówienia innych technik „zaciemniania” kanału.

4 Whitenoise – protokół kontroli transmisji w ukrytych kanałach informacyjnych TCP/IP

4.1 Podstawowe założenia

Celem stworzenia protokołu Whitenoise jest zapewnienie pełnej kontroli nad wymianą danych w ukrytych, sieciowych kanałach informacyjnych. Dotąd przedstawione rozwiązania techniczne nie zapewniają skuteczności dostarczenia danych, przez co ich zastosowanie praktyczne jest niewielkie. Dlatego też w czasie projektowania protokołu przyjęto, iż powinien on realizować następujące mechanizmy:

- Nawiązanie i zamknięcie połączenia pomiędzy dwoma komputerami – podział stron komunikacji na klienta i serwer.

- Dwukierunkową wymianę danych.
- Detekcję oraz korekcję błędów transmisji.
- Możliwość równoczesnej i niezależnej komunikacji z wieloma komputerami naraz.

Biorąc pod uwagę specyfikę zastosowania tego protokołu, powinien on wspierać takie rozwiązania jak:

- Rozróżnianie prawidłowych pakietów protokołu od danych losowych.
- Wsparcie dla szyfrowania danych.
- Wsparcie dla różnych rodzajów steganografii sieciowej.

Implementacja protokołu Whitenoise powinna wspierać wszystkie obecnie znane mechanizmy ukrywania danych w pakietach TCP/IP, w celu zminimalizowania możliwości wykrycia ukrytego kanału, osadzając dane w pakietach już istniejących – legalnych połączeń sieciowych, nie generując żadnego dodatkowego ruchu.

4.2 Testy poprawności oraz efektywności działania protokołu

Protokół i aplikacja go realizująca zostaną uznane za poprawnie działające, gdy bezbłędnie przejdą następujące testy:

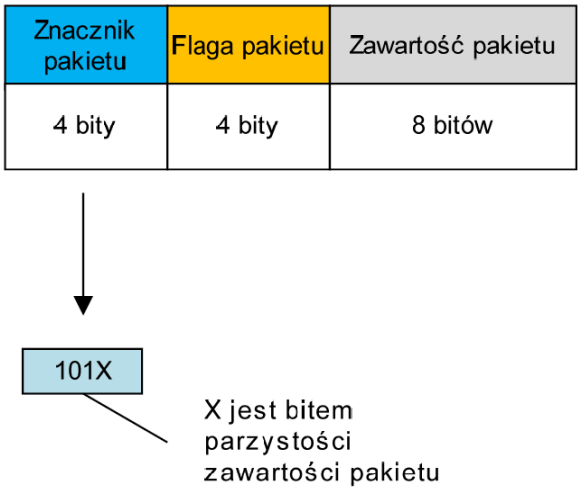
- Udane przesłanie pomiędzy dwoma komputerami w warunkach laboratoryjnych pliku o wielkości 100 MB wypełnionego losowymi danymi. Test zostanie uznany za pomyślny, jeżeli sumy kontrolne MD5 pliku oryginalnego i przesłanego będą identyczne.
- Udane przesłanie pomiędzy dwoma komputerami w warunkach rzeczywistych (publiczna sieć IP) pliku o wielkości 20 MB wypełnionego losowymi danymi. Test zostanie uznany za pomyślny, jeżeli sumy kontrolne MD5 pliku oryginalnego i przesłanego będą identyczne. W przypadku tego testu ilość danych jest mniejsza, gdyż ilość danych realnych transmisji internetowych, wymagana do przesłania 100 MB ukrytych danych może być trudna do osiągnięcia. W trakcie przebiegu tego testu na łączu sieciowym wprowadzane będą sztuczne zakłócenia (straty pakietów) mające za zadanie przetestować mechanizmy detekcji i korekcji błędów.
- Równoczesne przesłanie w obu kierunkach dwóch różnych 10 MB plików z losową zawartością w warunkach rzeczywistych (publiczna sieć IP). Test zostanie uznany za pomyślny, jeżeli sumy kontrolne MD5 pliku oryginalnego i przesłanego będą identyczne. Test ten ma za zadanie sprawdzić poprawność mechanizmów transmisji dwukierunkowej.
- Równoległe przesłanie dwóch 10 MB plików z losową zawartością poprzez publiczną sieć IP na dwa różne komputery. Test zostanie uznany za pomyślny, jeżeli sumy kontrolne obu plików przesłanych i oryginalnych będą identyczne. Test ten ma za zadanie sprawdzić prawidłowe działanie transmisji równoległych.

Po uznaniu protokołu za funkcjonujący poprawnie, sprawdzona zostanie jego efektywność, badając następujące wskaźniki:

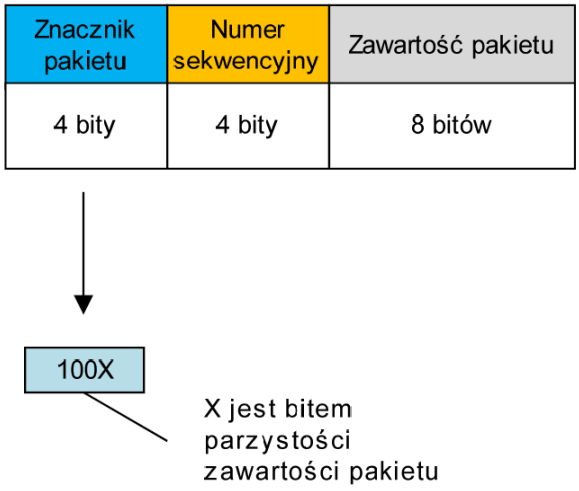
- Stosunek osiąganej przepustowości transmisji ukrytej do przepustowości transmisji jawnej – z podziałem na poszczególne mechanizmy ukrywania danych w pakietach.
- Średnią ilość danych, którą można przesłać poprzez ukryty kanał informacyjny w trakcie przeciętnych czynności wykonywanych przez użytkownika – przeglądanie stron internetowych, sprawdzenie poczty, itp. – z podziałem na scenariusz czynności oraz mechanizmy ukrywania danych w pakietach.
- Procentowe obciążenie procesora przez moduł Whitenoise w trakcie transmisji danych w zależności od przepustowości transmisji.
- Możliwości i prawdopodobieństwo wykrycia istnienia ukrytego kanału za pomocą analizy ciągów generowanych wartości pól pakietów.

4.3 Rodzaje pakietów

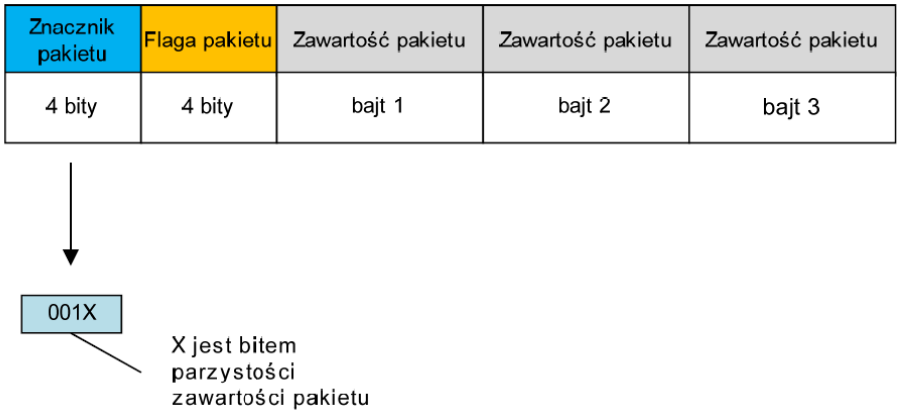
Wymiana danych w Whitenoise opiera się na 2 typach pakietów o stałej długości – pakietów sterujących i pakietów danych. Każdy z tych pakietów może wystąpić w rozmiarze 16 lub 32 bitowym - standardowych rozmiarach danych jakie w większości przypadków można jednorazowo ukryć w pojedynczym pakiecie TCP/IP. Schematy tych pakietów przedstawione są na Rys. 10, 11, 12 i 13. Zawartości pakietów układane są formacie Little-Endian – od lewej do prawej – od bitu najmłodszego do najstarszego. W tej też postaci transmitowane są przez sieć.



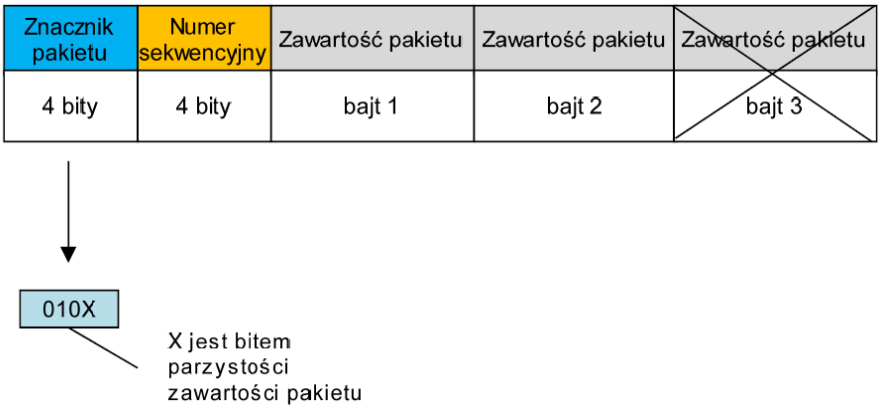
Rys. 10 16 bitowy pakiet sterujący Whitenoise



Rys. 11 16 bitowy pakiet danych Whitenoise



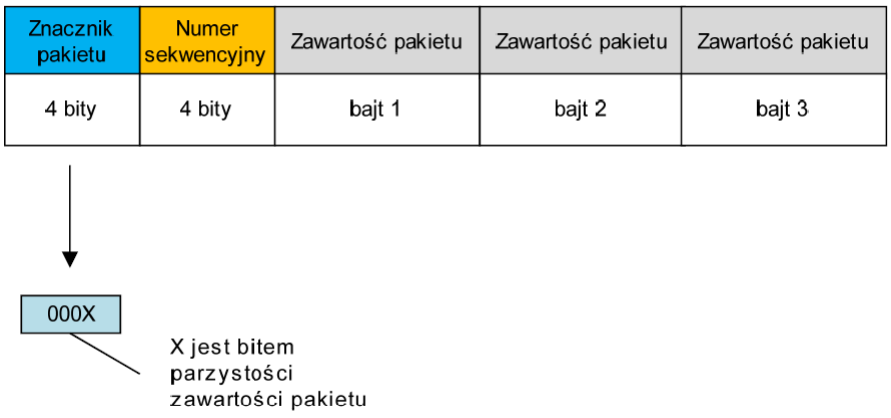
Rys. 12 32 bitowy pakiet sterujący Whitenoise



Rys. 13 32 bitowy pakiet danych Whitenoise

Znaczenie poszczególnych pól pakietów jest następujące:

- **Znacznik pakietu** - wskazuje rozmiar i typ pakietu. To oraz dodatkowy bit parzystości umożliwiają łatwe rozróżnienie czy otrzymane przez sieć dane są prawidłowym pakietem protokołu, czy nie.
- **Flaga Pakietu / Numer sekwencyjny** – W przypadku gdy pakiet jest pakietem sterującym - flaga określa funkcję jaką pełni pakiet (np. inicjującą połączenie). W przypadku, gdy pakiet jest pakietem danych - zawartość tego pola jest liczbą będącą numerem sekwencyjnym bajtu przesyłanego w danym pakiecie. Gdy w pakiecie znajduje się więcej bajtów danych, numer ten wskazuje na bajt pierwszy (numery bajtów kolejnych w pakiecie wyznaczane są analogicznie)
- **Zawartość pakietu** – dane transportowane przez pakiet. W momencie, gdy pakiet jest pakietem sterującym pole to zawiera dane określone przez aktualnie realizowaną funkcję. Ze względu na mocno ograniczone rozmiary pakietów, nie ma tutaj pola określającego ile bajtów aktualnie jest przesyłanych. W przypadku pakietów 16 bitowych problem ten nie występuje, gdyż tam można przesłać jedynie jeden bajt. W przypadku pakietów 32 bitowych można przesłać aż 3 bajty za jednym razem. Dlatego też wprowadzono 2 podtypy 32 bitowych pakietów danych, różniące się znacznikiem. W pierwszym (Rys. 13) przyjmuje się, że transmitowane są jedynie 2 bajty (trzeci jest wartością losową i nie jest brany pod uwagę). W drugim przypadku (Rys. 14) przyjmuje się, iż transmitowane są wszystkie trzy bajty. Nie ma wersji pakietu przesyłającego tylko jeden bajt. Jeżeli zachodzi taka potrzeba - używa się pakietu 16 bitowego.



Rys. 14 32 bitowy pakiet danych Whitenoise, transportujący 3 bajty

4.4 Flagi pakietów sterujących

Pakiety sterujące zawierają 4 bitowe pole flagi określające funkcję jaką pełni dany pakiet. Poniżej znajduje się tabela możliwych do użycia flag wraz z krótkimi ich opisami. Dokładne opisy funkcji i mechanizmów działania poszczególnych typów pakietów znajdują się w dalszej części pracy.

Tab. 1 Flagi pakietów sterujących Whitenoise

Wartość flagi (szesnastkowo)	Identyfikator flagi	Komentarz
0x01	**INIT**	Inicjacja połączenia
0x02	**INIT_ACK**	Potwierdzenie inicjacji połączenia
0x03	**INIT_ACK2**	Drugie potwierdzenie inicjacji połączenia
0x04	**NEW_BLOCK**	Rozpoczęcie nowego bloku danych
0x05	**NEW_BLOCK_ACK**	Potwierdzenie rozpoczęcia nowego bloku danych
0x06	**DATA_ACK**	Potwierdzenie odebrania danych
0x07	**CLOSE**	Zamknięcie połączenia
0x08	**CLOSE_ACK**	Potwierdzenie zamknięcia połączenia

4.5 Mechanizmy protokołu Whitenoise

4.5.1 Kontrola poprawności pakietu – bit parzystości

W celu podstawowej kontroli poprawności pakietu ostatni bit znacznika pakietu jest bitem parzystości. Wartość tego pola wynosi 1 jeżeli liczba jedynek w całym pakiecie jest nieparzysta, 0 zaś gdy liczba ta jest parzysta. Bit parzystości wyznacza się z całego pakietu. W przypadku pakietów zajmujących mniej miejsca niż ustalony rozmiar nośnika (np. pakiety 16 bitowe osadzone w polach 32 bitowych) bit parzystości wyznacza się z całego rozmiaru nośnika. Przyjmuje się, iż przy liczeniu parzystości bitów, bit kontroli parzystości w pakiecie ma wartość 0.

4.5.2 Nawiązanie połączenia

Do nawiązania połączenia pomiędzy obiema stronami komunikacji niezbędna jest wymiana pomiędzy nimi 3 pakietów. Pierwszy z nich – pakiet z ustawioną flaga INIT wysyła strona inicjująca połączenie (klient). W odpowiedzi strona nasłuchująca (serwer) powinna potwierdzić inicjację połączenia odsyłając pakiet INIT_ACK. Połączenie uznaje się za nawiązane, gdy strona inicjująca połączenie po odebraniu INIT_ACK odeśle do serwera pakiet INIT_ACK2. Trójfazowe nawiązanie połączenia, mimo iż nie jest ściśle wymagane do wymiany wszystkich niezbędnych informacji, zostało wprowadzone na wypadek gdyby okazało się, iż odebrany pakiet INIT_ACK nie został wysłany celowo przez klienta. W niektórych przypadkach istnieje pewne prawdopodobieństwo, iż losowe dane sieciowe mogą zostać błędnie zinterpretowane jako pakiet protokołu Whitenoise – szczególnie tyczy się to ukrywania danych w polach 16 bitowych, gdzie istnieje jedynie 65536 różnych ich wartości. Ściśle określony diagram stanów protokołu (opisany później) minimalizuje ryzyko takich zakłóceń.

Po wymianie wszystkich trzech pakietów połączenie uznaje się za nawiązane.

W momencie nawiązywania połączenia największym problemem staje się wybór dostępnych kanałów komunikacji – u podstaw architektury protokołu Whitenoise leży założenie, iż powinien on działać niezależnie od zastosowanego mechanizmu ukrywania i transmisji danych w pakietach sieciowych. Jako, iż każda ze stron komunikacji może wspierać lub wybrać inne kanały, niezbędne jest dwustronne ich wynegocjowanie. Odbywa się to dzięki zawartości pakietów INIT oraz INIT_ACK w której każda ze stron umieszcza informacje o obsługiwanych przez nią mechanizmach. Dane te zapisane są na 8 bitach tak by nawiązanie połączenia było możliwe również z użyciem pakietów 16 bitowych. Każdy z 8 bitów danych odpowiada za wskazanie jednej z dostępnych metod komunikacji. Włączony bit (1) wskazuje, iż dana metoda jest obsługiwana, bit wyłączony (0) wskazuje, iż dana strona komunikacji jej nie wspiera. Bity zarezerwowane powinny mieć wartości losowe.

Rys. 15 przedstawia strukturę zawartości pakietów INIT oraz INIT_ACK oraz znaczenie poszczególnych ich bitów. Zawartość pakietów z flagą INIT_ACK2 powinna być losowa.

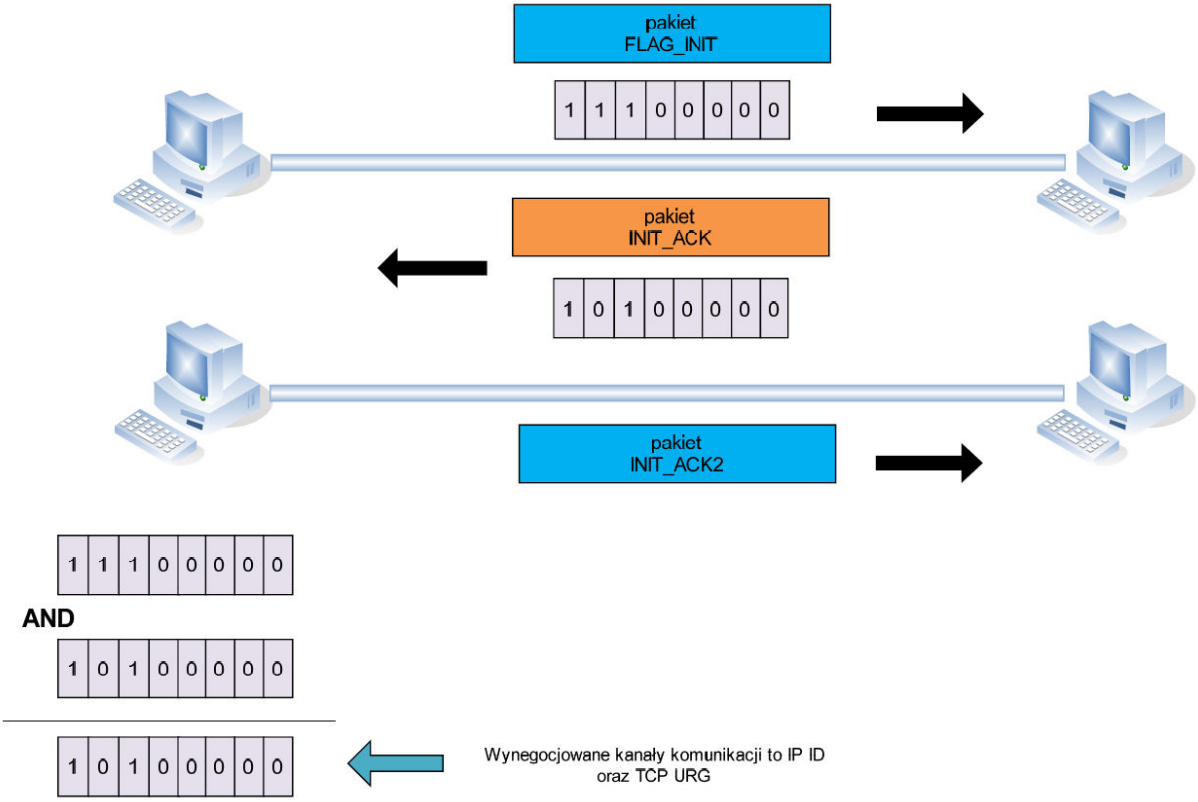
Bit 0	Bit 1	Bit 2	Bit 3	Bit 4	Bit 5	Bit 6	Bit 7
IP ID	TCP ISN	TCP URG	zarezerwowane	zarezerwowane	zarezerwowane	zarezerwowane	zarezerwowane

Rys. 15 Zawartość pakietów INIT oraz INIT_ACK

Dysponując informacjami o mechanizmach obsługiwanych przez strony komunikacji, każda z nich porównuje listę swoją i otrzymaną oraz wyznacza z nich część wspólną. Tak uzyskana lista przyjmowana jest za listę mechanizmów używanych do komunikacji obu stron.

W czasie nawiązywania połączenia (gdy sposoby komunikacji nie są jeszcze wynegocjowane), strona inicjująca połączenie zobowiązana jest wysyłać pakiety FLAG_INIT wszelkimi obsługiwanymi przez nią kanałami komunikacji po kolei tak długo, aż nie otrzyma odpowiedzi.

Diagram obrazujący nawiązanie połączenia demonstruje Rys. 16.



Rys. 16 Diagram nawiązania połączenia

4.5.3 Transmisja danych

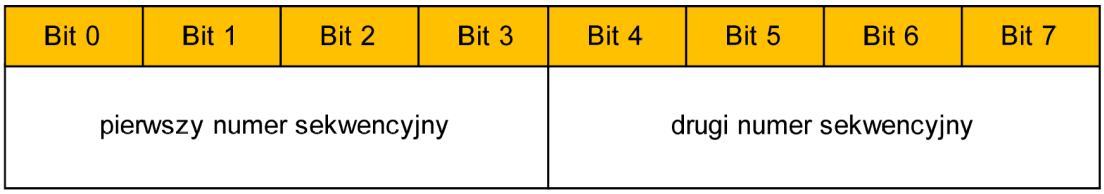
Ze względu na ograniczenie polegające na tym, iż pole w pakiecie danych zawierające numer sekwencyjny przesyłanego bajtu ma rozmiar jedynie 4 bitów, niezbędne było zastosowanie grupowania danych w bloki o rozmiarach 16 bajtów. Umożliwia to zastosowanie mechanizmów detekcji zagubionych pakietów z danymi oraz ich retransmisję. Każda ze stron komunikacji posiada dwie niezależne kolejki dla odbierania i wysyłania danych umożliwiając transmisję dwukierunkową.

Rozpoczęcie transmisji następuje w momencie, gdy jakakolwiek ze stron, posiadająca dane do przesłania, wyśle pakiet sterujący z flagą NEW_BLOCK. Powinien on zostać potwierdzony przez stronę drugą pakietem z flagą NEW_BLOCK_ACK. Po odebraniu potwierdzenia dana stacja rozpoczyna wysyłanie pakietów z danymi, numerując każdy przesyłany bajt.

Pierwsze 4 bity zawartości pakietu NEW_BLOCK powinny zawierać ostatni numer sekwencyjny bajtu przesyłanego w danym bloku. Bajty numerowane są począwszy od zera, tak więc ustawienie tej wartości na 15 wskazuje iż w danym bloku przesłanych zostanie 16 bajtów.

Zawartość pakietu NEW_BLOCK_ACK powinna być losowa.

Każdy bajt danych odebrany przez odbiorcę musi zostać potwierdzony. Odbywa się to za pomocą pakietów sterujących z flagą DATA_ACK. W zawartości pakietu umieszcza się numer sekwencyjny potwierdzanego bajtu. Jako, że po zapisaniu tam numeru sekwencyjnego pozostają wolne 4 bity zaimplementowana została metoda potwierdzania większej ilości danych naraz. Jeżeli pakiet DATA_ACK potwierdza jedynie jeden bajt, ostatnie 4 bity pakietu zostają ustawione na wartość 0. Jeżeli natomiast wartość ta jest niezerowa, wraz z poprzednia tworzą one przedział numerów sekwencyjnych bajtów odebranych poprawnie. Format zawartości pakietów DATA_ACK ilustruje Rys. 17

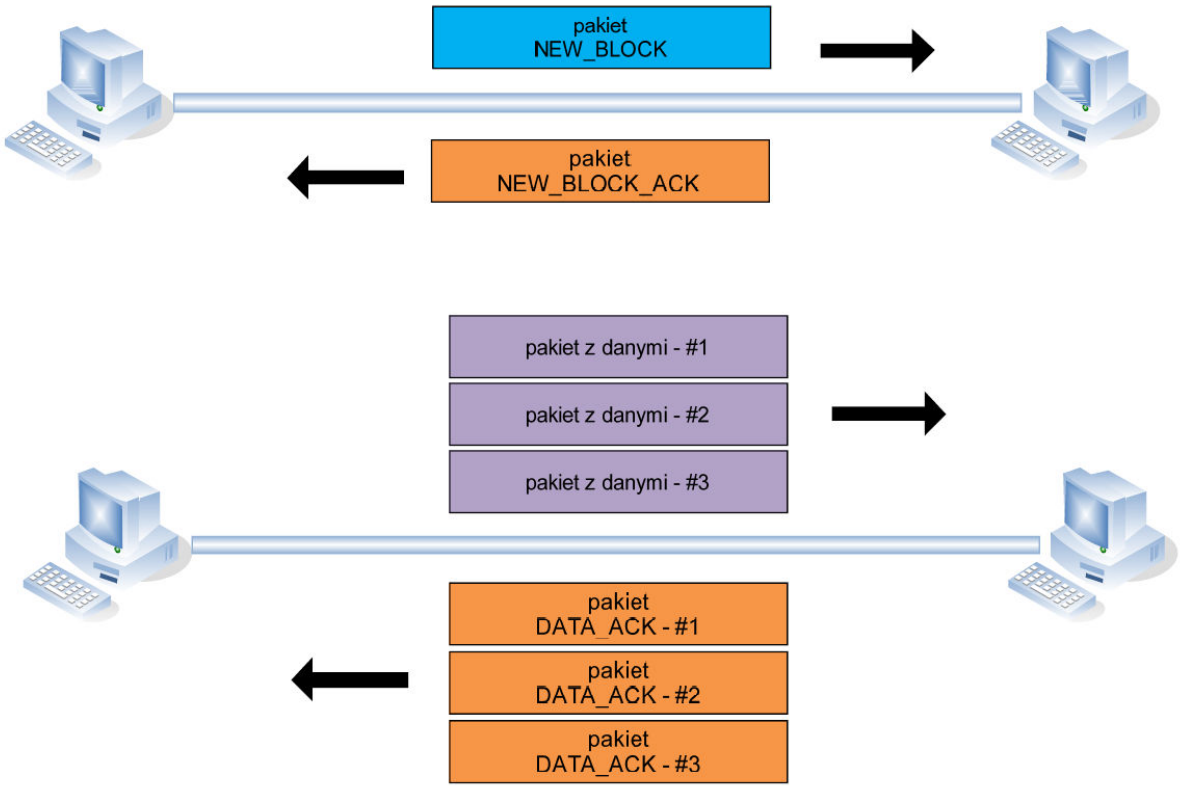


Rys. 17 Bajt zawartości pakietu DATA_ACK

Należy zwrócić uwagę, iż przedział bajtów odebranych poprawnie może być potwierdzony jedynie wtedy, gdy jest on kompletny. Załóżmy sytuację w której odebrane zostały bajty o numerach sekwencyjnych 5,6,7,8,9 – wtedy wystarczy potwierdzić je pakietem DATA_ACK z umieszczonymi w zawartości liczbami 5 oraz 9. Niemożliwe to będzie w przypadku, gdy np. zostaną odebrane bajty 5,6,8,9 – gdyż nie układają się one liniowo w przedział. W tym wypadku konieczne będzie potwierdzenie ich za pomocą 4 osobnych pakietów DATA_ACK.

Potwierdzanie odebrania danych w protokole Whitenoise jest asynchroniczne, to znaczy iż strona wysyłająca dane nie zatrzymuje się po wysłaniu każdego bajtu danych czekając na jego potwierdzenie, tylko wysyła je z maksymalną możliwą szybkością jednocześnie monitorując czy w międzyczasie nie pojawiło się potwierdzenie odebrania któregoś z nich. Po wysłaniu wszystkich danych z bloku czeka przez pewien ustalony czas na potwierdzenie odebrania wszystkich bajtów a po jego upływie ponownie – do skutku - retransmituje niepotwierdzone bajty. Po odebraniu potwierdzeń wszystkich wysłanych bajtów rozpoczęty może zostać kolejny blok.

Pełna procedura wysyłania i potwierdzania danych przedstawiona jest na Rys. 18



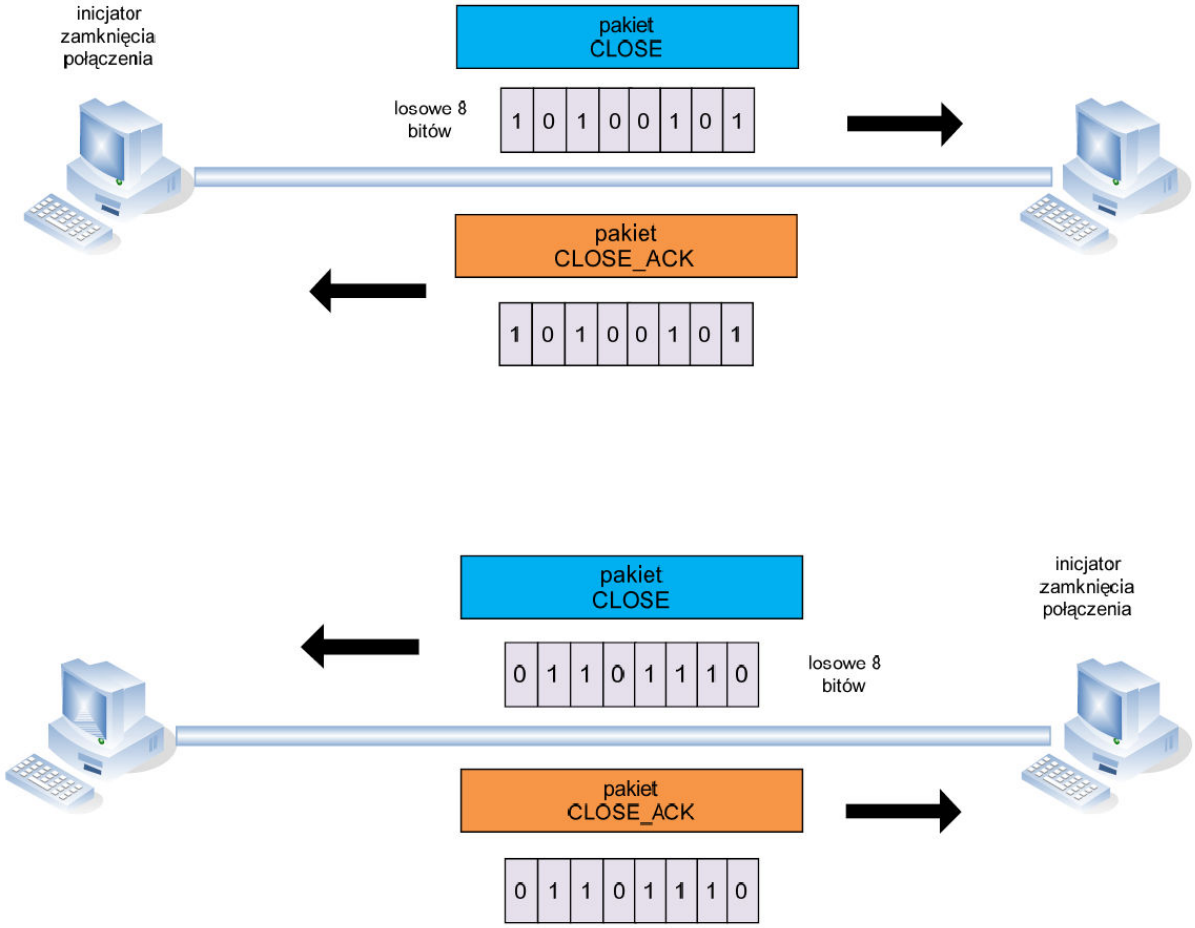
Rys. 18 Procedura wysyłania i potwierdzania danych

4.5.4 Zamknięcie połączenia

Zamknięcie połączenia zostaje jawnie przeprowadzone przez użytkowników protokołu wtedy, gdy nie mają już do przesłania żadnych danych i chcą ten fakt zakomunikować drugiej stronie. Podobnie jak w protokole TCP może dotyczyć to jedynie poszczególnej strony komunikacji – wtedy może ona tylko odbierać dane. Zamknięcie połączenia przez obie strony komunikacji definitywnie kończy połączenie. Procedura zamknięcia połączenia może zostać przeprowadzona jedynie wtedy, gdy wszystkie, aktualnie rozpoczęte bloki danych zostaną poprawnie przesłane.

Zamknięcie połączenia przez każdą ze stron wygląda identycznie i wymaga wymiany 2 pakietów – CLOSE oraz CLOSE_ACK. Strona inicjująca zamknięcie wysyła pakiet z ustawioną flagą CLOSE i losową zawartością (8 bitów). Druga strona komunikacji w odpowiedzi odsyła pakiet z flagą CLOSE_ACK i dokładnie taką samą zawartością.

Schemat zamykania połączenia przez każdą ze stron przedstawiony jest na Rys. 19.



Rys. 19 Zamknięcie połączenia

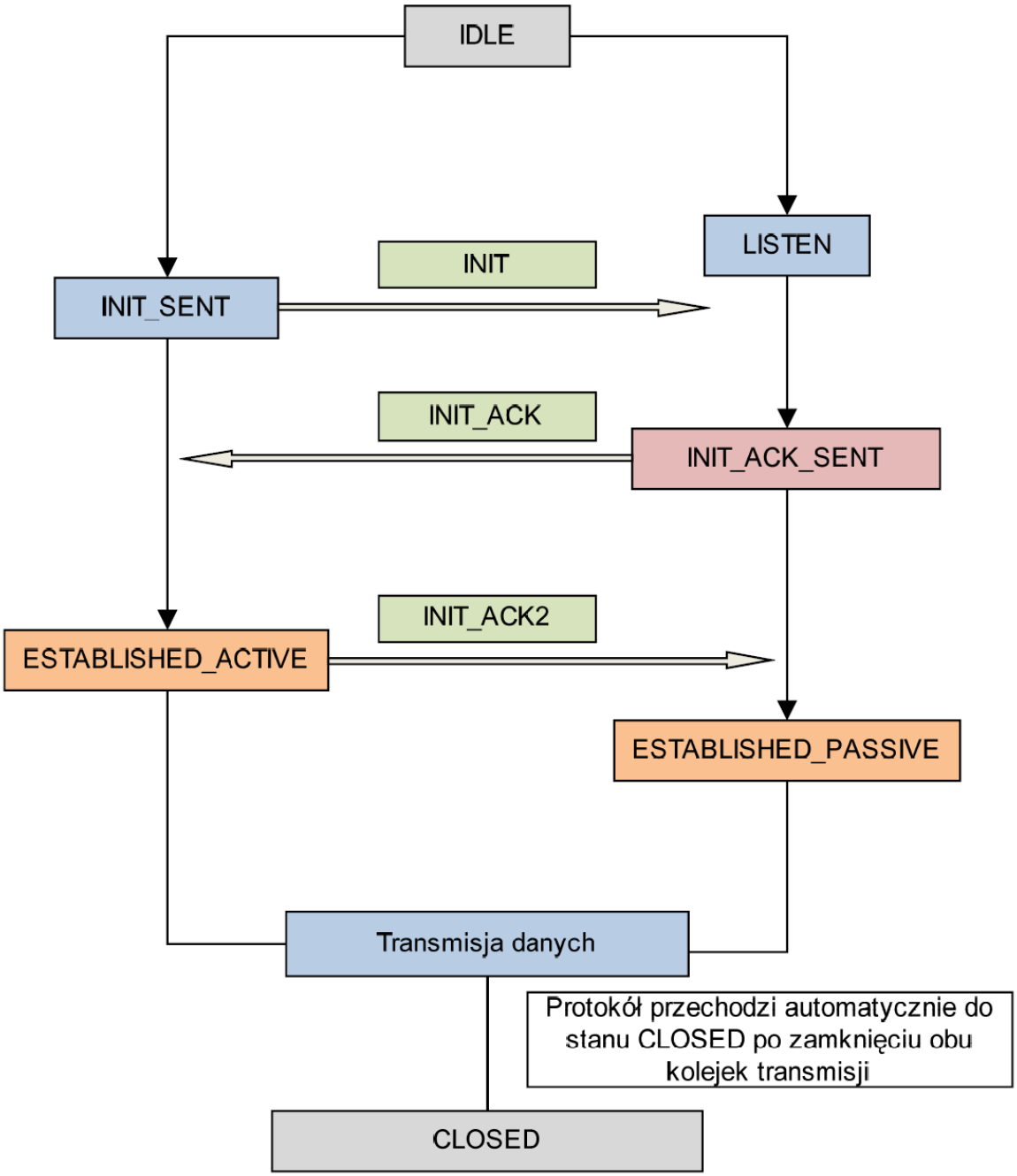
4.5.5 Mechanizmy detekcji oraz korekcji błędów

Jedną z cech sieci komputerowych jest fakt, iż niemal każdy transmitowany pakiet może zostać uszkodzony lub zagubiony, zazwyczaj z powodu zakłóceń na łączu lub wadliwie działających urządzeń pośredniczących w transmisji. Projektując mechanizmy protokołu Whitenoise należało również liczyć się z tym, iż w każdym momencie pakiet TCP/IP za pomocą którego przesyłane są dane może zostać utracony. Zaimplementowany mechanizm detekcji oraz korekcji błędów jest nieco podobny do tego stosowanego w protokole TCP i opiera się na szeregu stanów w jakich może się znaleźć protokół. Diagram ten przedstawiony jest na Rys. 20.

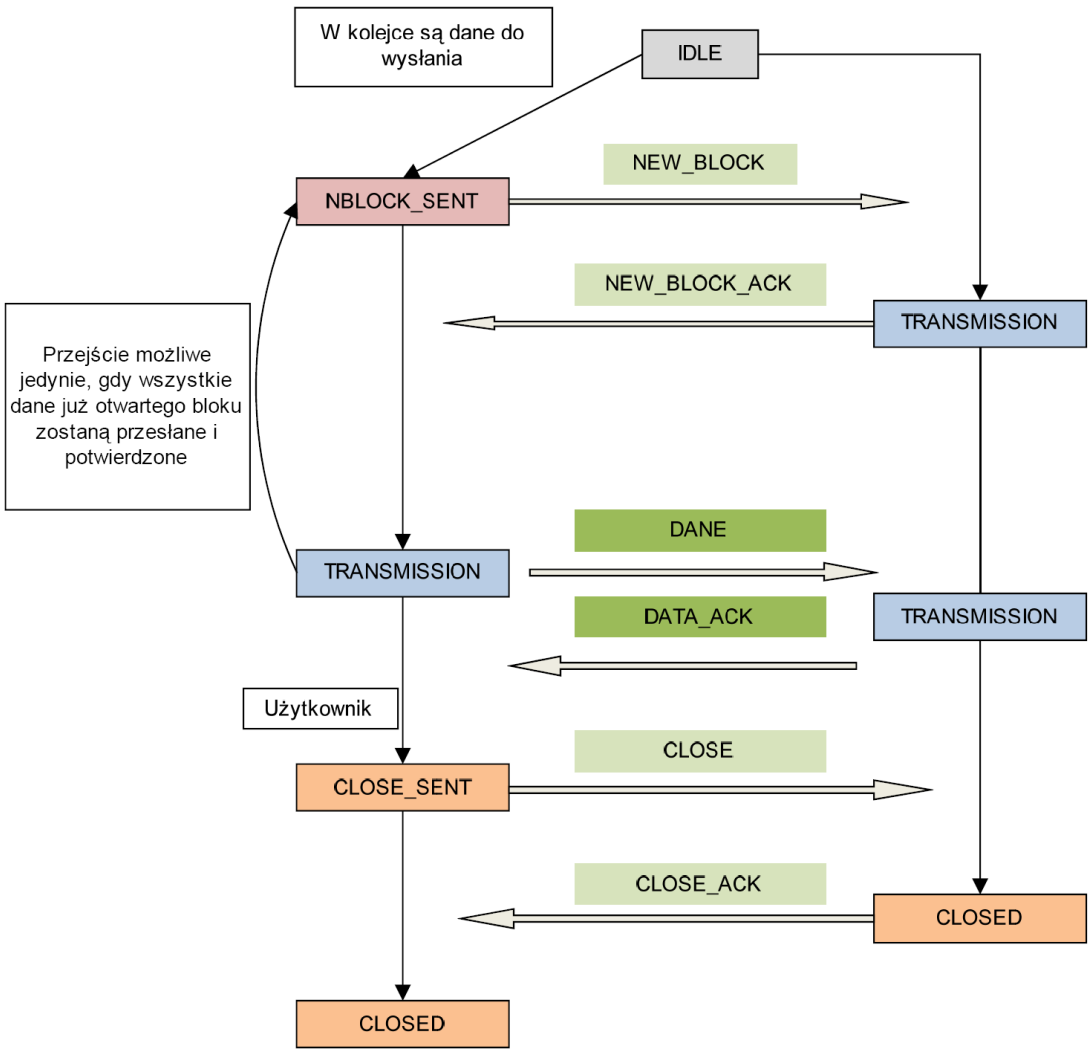
Tuż na początku pracy protokół znajduje się w stanie IDLE. Z tego stanu użytkownik próbujący nawiązać komunikację zmienia jego stan na LISTEN – jeżeli dana końcówka sieciowa ma pełnić rolę serwera lub INIT_SENT – jeżeli dana końcówka ma zainicjować nawiązanie połączenia, jednocześnie zakolejkowany zostanie do wysłania pakiet INIT.

Od tego momentu każdy wysłany lub odebrany pakiet Whitenoise zmienia bieżący stan protokołu zgodnie ze schematem. Po każdym wysłaniu pakietu na który oczekiwana jest odpowiedź ustawiany jest specjalny licznik zliczający pakiety TCP/IP przychodzące od odbiorcy. Za przekroczony czas oczekiwania zostaje uznana sytuacja w której zostanie odebrana odpowiednia ilość pakietów (domyślnie 15) TCP/IP w których mogły zostać osadzone pakiety Whitenoise, jednak nie zostały. Jest to jedyna alternatywa dla stosowanych zazwyczaj rozwiązań opartych o mierzenie czasu jaki upłynął od czasu wysłania pakietu. Takie rozwiązanie tutaj nie mogło być jednak zastosowane gdyż ukrywając dane w już istniejących połączeniach sieciowych nie mamy wpływu na ich przebieg – gdy np. pakiety są wysyłane z częstotliwością kilku na godzinę. W momencie gdy limit został przekroczony, bieżący status protokołu pozwala na ocenienie który z pakietów został stracony i ponowne jego wysłanie. Żadna ze stron nie ma możliwości sprawdzenia czy stracony został pakiet przez nią wysłany czy pakiet będący na niego odpowiedzią, strona która nie doczekała się odpowiedzi ponawia wysłanie pakietu a strona druga – nawet jeżeli już wysłała swoją odpowiedź wcześniej musi na nią odpowiedzieć raz jeszcze. Każdy odebrany pakiet, który nie pasuje do bieżącego stanu protokołu musi zostać odrzucony.

Opisane wyżej mechanizmy dotyczą wymiany pakietów stertujących protokołu. Po nawiązaniu połączenia i osiągnięciu stanu ESTABLISHED_ACTIVE lub ESTABLISHED_PASSIVE rozpoczyna się wymiana danych opisana w rozdziale 4.5.3. Każdy z niezależnych mechanizmów odpowiedzialnych kolejno za wysyłanie i odbieranie danych posiada własny rejestr stanu – zmieniający się zgodnie z diagramem przedstawionym na Rys. 21 – oraz liczniki przekroczenia czasu oczekiwania działające identycznie jak te opisane wcześniej. Detekcja i korekcja zagubionych pakietów z danymi następuje po wysłaniu wszystkich pakietów. Po wysłaniu ostatniego zostaje ustawiony licznik oczekiwania, jeżeli przed przekroczeniem jego limitu zostaną odebrane potwierdzenia otrzymane od drugiej strony transmisji zostaje on wyłączony i blok uznany zostaje na przesłany poprawnie. W przeciwnym wypadku – gdy nie zostały odebrane wszystkie potwierdzenia, zagubione pakiety transmitowane są ponownie – do skutku.



Rys. 20 Diagram stanów protokołu Whitenoise



Rys. 21 Diagram stanów mechanizmu wymiany danych

4.5.6 Kryptografia i bezpieczeństwo w Whitenoise

W punkcie tym zostaną omówione mechanizmy stosowane w protokole Whitenoise, pozwalające na ukrycie przesyłanych pakietów protokołu pod pozorem pseudolosowych danych.

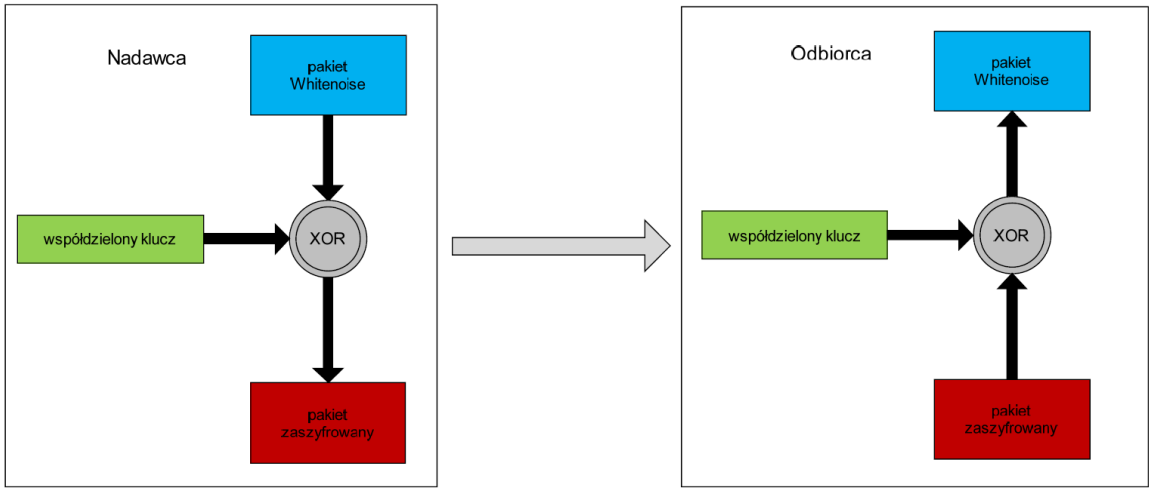
Kryptografia

Jak zostało to wspomniane w dokumentacji NUSHU [16], w wypadku pakietów o bardzo małych rozmiarach (w naszym wypadku 16 i 32 bitów) stosowanie popularnych metod kryptograficznych jest mocno utrudnione. W przypadku szyfrów strumieniowych, kodowanie tak małej ilości danych za jednym razem nie zapewnia niemal żadnego bezpieczeństwa (klucze mają rozmiar 16/32 bit), zaś w przypadku szyfrów blokowych minimalny rozmiar kodowanego bloku to 64 bit – zupełnie dla nas nieprzydatny.

W przypadku ukrytego kanału zadaniem kryptografii jest nie tyle zapewnienie mocnego bezpieczeństwa przesyłanym danym (te mogą zostać zaszyfrowane na innym poziomie) co takie przekształcenie pakietów, aby wyglądały na zwykłe, pseudolosowe liczby.

Sposób szyfrowania danych w Whitenoise został zaczerpnięty z NUSHU i polega na XOR-owaniu pakietów stałym, współdzielonym przez obie strony komunikacji kluczem. Sposób użycia klucza jest modyfikacją w stosunku do NUSHU, gdzie dla każdego pakietu TCP/IP klucz był modyfikowany w oparciu o adresy i porty nadawcy i odbiorcy, zapewniając większą losowość. Przyjęte w Whitenoise rozwiązanie ma swoje wady – stały klucz może powodować iż takie same pakiety będą po zaszyfrowaniu wyglądać identycznie – przecząc „udawanej" pseudolosowości. Ma również zalety – uniezależnia pakiety Whitenoise od konieczności osadzania w konkretnym protokole (tutaj TCP) jednocześnie wprowadzając po drobnej modyfikacji możliwość użycia wektorów inicjalizacyjnych (IV) zapewniających kodowanie każdego pakietu innym kluczem. Możliwe jest też zaimplementowanie cyklicznej negocjacji nowych kluczy.

Rys. 22 przedstawia schemat szyfrowania pakietów Whitenoise.



Rys. 22 Schemat szyfrowania pakietów

Losowość zawartości pakietów

Jak zostało wspomniane powyżej, użycie szyfrowania metodą XOR i współdzielonego klucza może sprawić, iż próba ukrycia pakietów Whitenoise pod postacią pseudolosowych liczb nie powiedzie się. Identyczne pakiety, po zaszyfrowaniu, za każdym razem będą wyglądać identycznie i dla osoby monitorującej ruch sieciowy takie anomalie staną się jasną wskazówką, iż ktoś manipuluje danym protokołem sieciowym. Aby choć w części zniwelować ten efekt, Whitenoise posiada możliwość zawierania w swoich pakietach danych losowych, tak aby dwa pakiety tego samego typu i zawartości, wyglądały inaczej. Mechanizm ten wykorzystuje fakt, iż czasem zawartość pakietu Whitenoise (zwłaszcza pakietów sterujących) nie jest wykorzystywana. Dzięki temu w miejscach tych można umieścić bity losowe, utrudniające osobom postronnym znalezienie jakiegokolwiek wzorca. Identycznie ma się sprawa w przypadku, gdy chcemy umieścić 16 bitowy pakiet Whitenoise w 32 bitowym polu nagłówka TCP/IP – wtedy wystarczy ostatnie 2 bajty pola wypełnić wartościami losowymi.

Innym, bardzo ważnym mechanizmem jest możliwość cyklicznego umieszczania w pakietach TCP/IP zamiast pakietu Whitenoise prawdziwej wartości wygenerowanej przez system operacyjny dla tego pola. Umożliwia to mechanizm rozróżniania prawidłowych pakietów za pomocą znaczników i bitów parzystości. W momencie, gdy Whitenoise w danym momencie nie posiada w kolejce żadnego pakietu do wysłania, w modyfikowanym polu nagłówka TCP/IP pozostawiona zostaje jego oryginalna wartość. Pod warunkiem, iż nie może ona zostać błędnie zinterpretowana jako pakiet Whitenoise – w tym wypadku modyfikowany jest jej najmłodszy bit tak, aby kontrola bitu parzystości u odbiorcy go odrzuciła. Takie przeplatanie danych rzeczywistych i należących do kanału ukrytego może skutecznie utrudnić próby detekcji ukrytego kanału za pomocą analizy rozkładu statystycznego.

W pakietach protokołu Whitenoise losowe dane mogą być umieszczane w następujących miejscach:

- Bity zarezerwowane zawartości pakietów sterujących z flagami FLAG_INIT oraz FLAG_INIT_ACK (obecnie 5 najmłodszych bitów).
- Cała 8 bitowa zawartość pakietu sterującego z flagą FLAG_INIT_ACK2.
- Cała 8 bitowa zawartość pakietu sterującego z flagą CLOSE.
- Ostatni bajt 32 bitowego pakietu danych ze znacznikiem 010 (transportujący jedynie 2 użyteczne bajty danych).
- Ostatnie 2 bajty 32 bitowej wartości w której osadzony jest pakiet 16 bitowy.

4.5.7 Mechanizmy ukrywania danych w pakietach TCP/IP

Mechanizm IP ID

Używany protokół: IP

Używane pole protokołu: ID

Rozmiar pola: 16 bitów

Zawartość pola: właściwa dla danej pary adresów IP pseudolosowa liczba, rosnąca sekwencyjnie wraz z ilością wysłanych pakietów. Służąca do ustalania kolejności pakietów IP w czasie ich fragmentacji i defragmentacji przez routery.

Wykrywalność: bardzo wysoka

Uwagi: pierwsza z metod zaimplementowana w Whitenoise. Używana do testów oraz uzyskiwania wysokich przepustowości ukrytych kanałów bez zwracania uwagi na wysoką wykrywalność.

Metoda opisana w: [15], [1], [13]

Mechanizm TCP ISN

Używany protokół: TCP

Używane pole protokołu: SEQ (w pakietach z flagą SYN lub SYN/ACK)

Rozmiar pola: 32 bity

Zawartość pola: pseudolosowa liczba będąca początkowym numerem sekwencyjnym w połączeniu protokołu TCP.

Wykrywalność: niska

Uwagi: wykrywalność w dużym stopniu zależy od użytej kryptografii zastosowanej na pakietach Whitenoise. Podstawowy mechanizm używany w tworzeniu ukrytych kanałów. W przeciwieństwie do NUSHU, implementacja Whitenoise nie utrzymuje tablic ze stanami połączeń tylko modyfikuje stos TCP/IP systemu operacyjnego tak, aby „fałszywe" numery ISN były uznawane za te wygenerowane przez system.

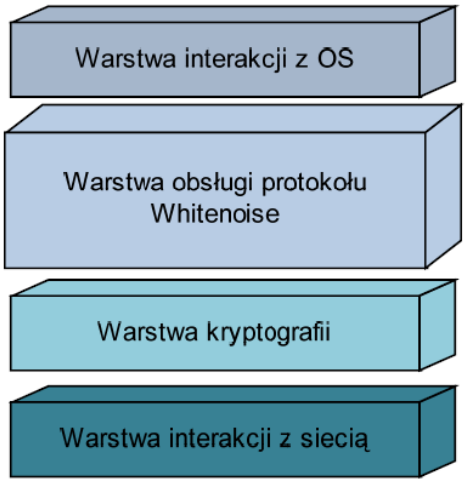
Metoda opisana w: [16], [13]

5 Implementacja protokołu Whitenoise w systemie Linux

Implementacja protokołu Whitenoise oraz wszystkich omówionych jego mechanizmów wykonana została w systemie Linux pod postacią ładowalnego modułu jądra systemu (wersji 2.6).

Dla efektywności oraz łatwej możliwości rozbudowy, moduł został podzielony na 4 warstwy (przedstawione na Rys. 23), każda realizująca inną funkcję:

1. Warstwa interakcji z systemem operacyjnym.
2. Warstwa obsługi protokołu Whitenoise.
3. Warstwa kryptografii.
4. Warstwa interakcji z siecią.



Rys. 23 Warstwowy model modułu Whitenoise

Warstwa interakcji z systemem operacyjnym odpowiedzialna jest za realizację wszystkich mechanizmów dotyczących inicjalizacji i kontroli poprawności działania modułu oraz komunikacji i udostępnienia interfejsów komunikacji z użytkownikiem.

Wszelka kontrola działań modułu oraz transmisja danych za jego pośrednictwem odbywa się korzystając z wirtualnego systemu plików *procfs*. Warstwa ta udostępnia w systemie operacyjnym plik */proc/whitenoise* za pomocą którego aplikacje przestrzeni użytkownika korzystając z wywołań funkcji *ioctl* dokonują konfiguracji, inicjalizacji i kontroli kanału informacyjnego, a następnie za pomocą funkcji *read/write* na deskryptorze otwartego pliku wymieniają dane z druga strona komunikacji. Po otwarciu pliku przez aplikację użytkownika i otrzymaniu jego deskryptora, w warstwie tej przydzielane jest nowe gniazdo, które po skonfigurowaniu służy do nawiązywania połączeń i wymiany danych z jednym konkretnym punktem docelowym. W obecnej implementacji moduł obsługuje 16 gniazd, z których każde może prowadzić niezależne połączenie poprzez ukryty kanał informacyjny.

Warstwa obsługi protokołu Whitenoise realizuje wszystkie opisane w niniejszej pracy mechanizmy dotyczące zarządzania połączeniami oraz wymianą danych. Również tutaj realizowane są algorytmy detekcji błędów (oparte o *timeouty*) i ich korekcji. Integralnym zadaniem tej warstwy jest również utrzymywanie rejestrów stanów połączenia i przekazywanie wymienianych danych i komunikatów do warstwy interakcji z systemem operacyjnym. Wymiana danych z sąsiadującymi warstwami odbywa się za pomocą 4 kolejek FIFO – 2 kolejek danych (wyjściowej i wejściowej) wymienianych z warstwą wyższą oraz 2 kolejek pakietów (wchodzących i wychodzących) wymienianych z warstwą niższą

Warstwa kryptografii odpowiedzialna jest za kodowanie pakietów opuszczających warstwę obsługi protokołu i dekodowanie pakietów przychodzących zanim zostaną przez nią przetworzone.

Warstwa interakcji z siecią, korzystając z interfejsu *Netfilter* w jądrze systemu Linux, monitoruje wszystkie pakiety IP i TCP odbierane i wysyłane przez system operacyjny. Dla każdego pakietu sprawdzane są adresy źródłowe i docelowe aby sprawdzić, czy należą one do komputerów w sieci prowadzących ukrytą transmisję sieciową. Jeżeli tak, wyszukiwane jest odpowiednie gniazdo połączenia i sprawdzana jest jego konfiguracja oraz kolejki pakietów. Jeżeli w kolejce oczekuje pakiet do wysłania, osadzany jest on w pakiecie TCP/IP za pomocą jednej z metod właściwych dla połączenia (wynegocjowanej przez obydwie strony komunikacji). Bardzo podobnie wygląda przypadek odebrania pakietu TCP/IP – jeżeli para adresów pasuje do maszyn prowadzących ukrytą komunikację, sprawdzane są wszystkie miejsca w których mogły zostać ukryte pakiety Whitenoise – jeżeli któryś został znaleziony, zostaje dodany do właściwej kolejki pakietów odebranych.

5.1 Korzystanie z modułu Whitenoise przez aplikacje

W niniejszym punkcie opisany został interfejs programistyczny (API) dla aplikacji pisanych w języku C, umożliwiający komunikację z modułem Whitenoise w jądrze systemu oraz nawiązywanie za jego pomocą połączeń ukrytymi kanałami.

5.1.1 Wymagane pliki nagłówkowe

Wszystkie niezbędne makra, zmienne i struktury danych konieczne do korzystania z modułu Whitenoise zawarte są w pliku nagłówkowym *whitenoise_client.h* znajdującym się w pakiecie zawierającym pliki źródłowe modułu.

5.1.2 Otwarcie i konfiguracja gniazda

Praca z modułem Whitenoise zaczyna się od otwarcia pliku */proc/whitenoise* – wymagane są prawa użytkownika *root*. Jeżeli takiego pliku nie ma w systemie oznacza to, iż moduł Whitenoise nie został załadowany do jądra systemu. Jeżeli funkcja open zamiast deskryptora pliku zwróciła nam błąd *errno* o kodzie EMFILE (*Too many open files*) oznacza to, iż plik */proc/whitenoise* został otwarty przez maksymalną ilość aplikacji naraz (domyślnie 16). W momencie otwarcia */proc/whitenoise* - danemu deskryptorowi pliku przypisywane jest nowe gniazdo oraz wszystkie niezbędne do prowadzenia połączenia struktury danych. Tak powstałe gniazdo należy skonfigurować za pomocą wywołań funkcji *ioctl*.

Pierwszą i najważniejszą opcją, jaką trzeba ustawić jest para adresów IP komputerów pomiędzy którymi prowadzić będziemy ukrytą transmisję – adresu lokalnego i adresu zdalnego. Adresem lokalnym jest adres IP naszego komputera, a adresem zdalnym adres IP komputera z którym tworzymy ukryty kanał. Konfiguracja ta odbywa się za pomocą dwóch

wywołań *ioctl* (zakładamy, że *fd* to zmienna zawierająca deskryptor otwartego pliku */proc/whitenoise*):

ioctl(fd, WN_IOCTL_SET_LOCAL_ADDR, inet_addr(“10.0.0.1”));

ioctl(fd, WN_IOCTL_SET_REMOTE_ADDR, inet_addr(“10.0.0.2”));

inet_addr jest funkcją systemową zamieniającą adres IP w postaci napisu na liczbę typu integer, zdefiniowana jest w pliku nagłówkowym <arpa/inet.h>

Drugą rzeczą niezbędną do ustawienia przed nawiązaniem połączenia jest wybór kanałów komunikacji (metod ukrywania pakietów Whitenoise w nagłówkach TCP/IP). Dokładne ich omówienie znajduje się w rozdziale 4.5.8 niniejszej pracy. Jeżeli ustawienia tego nie dokonamy osobiście, wybrane zostaną metody ustawione domyślnie w module.

Do ustawienia używanych kanałów służy struktura *wn_channels*. W poniższym przykładzie włączymy wszystkie obsługiwane przez moduł kanały:

struct wn_channels kanaly;

bzero(&kanaly, sizeof(struct wn_channels));

kanaly.ip_id=1;

kanaly.tcp_isn=1;

ioctl(fd, WN_IOCTL_SET_CHANNELS, &kanaly);

Analogicznie możemy pobrać z modułu listę aktualnie włączonych kanałów komunikacji:

struct wn_channels kanaly;

ioctl(fd, WN_IOCTL_GET_CHANNELS, &kanaly);

5.1.3 Nawiązanie połączenia

Podobnie jak w TCP/IP, nawiązanie połączenia może być aktywne (klient) lub pasywne (serwer). Z punktu widzenia aplikacji różnica jest bardzo niewielka:

Aby nawiązać połączenie aktywnie wywołujemy funkcję *ioctl* następująco:

ioctl(fd, WN_IOCTL_CONNECT);

Natomiast druga strona komunikacji, pełniąca rolę serwera powinna wywołać:

ioctl(fd, WN_IOCTL_LISTEN);

Od czasu wywołania WN_IOCTL_CONNECT rozpoczyna się procedura nawiązywania połączenia, która w zależności od wybranych kanałów komunikacji i ilości ruchu sieciowego pomiędzy komputerami może trwać od kilku sekund do nawet kilku godzin, a w naprawdę skrajnych przypadkach dni. Program użytkownika powinien co pewien ustalony czas

sprawdzać bieżący stan połączenia za pomocą wywołania WN_IOCTL_STATUS, podając jako argument wskaźnik do zmiennej typu *unsigned int*. Jeżeli zwrócona wartość będzie równa stałej WN_STATUS_ESTABLISHED_ACTIVE (dla klienta) lub WN_STATUS_ESTABLISHED_PASSIVE (dla serwera), oznacza to iż połączenie zostało nawiązane i można rozpocząć wymianę danych.

Przykład sprawdzenia statusu połączenia (dla klienta):

```
unsigned int  status;
ioctl(fd, WN_IOCTL_STATUS, &status);

if(status==WN_STATUS_ESTABLISHED)
        // połączenie nawiązane
```

5.1.4 Wymiana danych

Po nawiązaniu połączenia wymiana danych z drugą stroną komunikacji odbywa się poprzez zapisywanie oraz czytanie z deskryptora pliku za pomocą funkcji systemowych *read* oraz *write*. Ze względu na istnienie wewnętrznych kolejek danych Whitenoise udane zapisanie danych oznacza jedynie, iż zostały one przekazane modułowi do przetworzenia i wysłania, a nie iż zostały przesłane do odbiorcy. W przypadku całkowitego zapełnienia kolejki aplikacja zapisująca do deskryptora może zostać uśpiona do czasu zwolnienia się miejsca. W przypadku czytania z deskryptora zwracane są dane aktualnie dostępne w kolejce – funkcja *read* może zwrócić mniej bajtów niż zażądano. Jeżeli w kolejce nie ma aktualnie żadnych dostępnych danych, wywołanie *read* jest usypiane do czasu pojawienia się ich.

Po zapisaniu danych do deskryptora, za pomocą wywołania *ioctl* można sprawdzić ile bajtów znajduje się jeszcze w kolejce. Służy do tego wywołanie WN_IOCTL_OUTQUEUE_SIZE, zwracające ilość bajtów oczekujących na wysłanie (wliczając bajty znajdujące się w aktualnie transmitowanym bloku). Analogicznie WN_IOCTL_INQUEUE_SIZE zwraca liczbę bajtów dostępnych do pobrania (bez wliczania tych aktualnie transmitowanych w bloku). Oba wywołania jako parametr pobierają wskaźnik do zmiennej typu *unsigned int*.

Poniższy przykład demonstruje zapisanie danych do deskryptora oraz sprawdzenie, ile z nich zostało wysłanych w ciągu 10 sekund oraz ile przez ten czas odebrano.

```
unsigned int odebrano, wyslano;

write(fd, „test”, 5);
sleep(10);
ioctl(fd, WN_IOCTL_OUTQUEUE_SIZE, &wyslano);
ioctl(fd, WN_IOCTL_INQUEUE_SIZE, &odebrano);

printf(“Wyslano %d z 5 bajtow, odebrano %d bajtow\n”, 5-wyslano, odebrano);
```

5.1.5 Zamknięcie połączenia

W momencie, gdy jedna ze stron nie ma już więcej danych do przesłania, wywołuje na deskryptorze procedurę WN_IOCTL_CLOSE. Wywołanie to zamyka kolejkę wyjściową i sygnalizuje drugiej stronie transmisji zamknięcie połączenia. Po zamknięciu połączenia, wywołanie funkcji *write* na deskryptorze zwróci wartość 0, tak samo wywołanie funkcji *read* na deskryptorze po drugiej stronie połączenia (zakładając, że nie ma już żadnych oczekujących danych w kolejce odbiorcy). Po zamknięciu kolejki dana strona komunikacji może tylko czytać dane z deskryptora tak długo, aż druga strona połączenia nie skończy nadawać i analogicznie nie zamknie kolejki (i *read* zwróci nam wartość 0).

Niestety niemożliwe jest zakończenie działania aplikacji ani zamknięcie deskryptora pliku Whitenoise funkcją *close*. Spowodowane jest to faktem, iż po całkowitym zamknięciu deskryptora niszczone są wszystkie struktury zachowujące stan połączenia – druga strona połączenia nie zdąży zostać poinformowana o zamknięciu połączenia, ani też sama nie będzie mogła przeprowadzić zamknięcia i będzie oczekiwać bez końca na odpowiedź. Aby uniknąć tego typu sytuacji, po zamknięciu połączenia za pomocą WN_IOCTL_CLOSE i zakończeniu czytania danych z deskryptora, należy wywołać procedurę WN_IOCTL_WAIT. Usypia ona proces na okres czasu niezbędny do prawidłowej wymiany pakietów i upewnienia się, iż obie strony połączenia prawidłowo je zakończyły.

5.1.6 Obsługa błędów

Wywołanie każdej z funkcji systemowych: *open*, *ioctl*, *read*, *write* używanych do komunikacji z modułem Whitenoise może zakończyć się błędem. Identycznie jak przy innych tego typu wywołaniach w systemach klasy UNIX błąd sygnalizowany jest zwróceniem przez funkcję wartości mniejszej od 0 oraz ustawieniem odpowiedniego kodu błędu w zmiennej globalnej *errno*. Pomijając kody błędów, jakie mogą być generowanie przez sam system operacyjny (np. próba otwarcia nieistniejącego pliku albo brak praw dostępu), w Tab. 2 znajduje się wyszczególnienie kodów błędów generowanych przez moduł Whitenoise, wraz z ich krótkim omówieniem.

Tab. 2 Możliwe błędy funkcji systemowych

Funkcja	Tryb wywołania	Kod błędu	Komentarz
open	Nie dotyczy	EMFILE	Zbyt duża ilość jednocześnie otwartych deskryptorów pliku Whitenoise.
		ENOMEM	Błąd alokacji pamięci w Whitenoise. Zbyt mała ilość wolnej pamięci RAM.

read	Nie dotyczy	ENOMEM	Błąd alokacji pamięci w Whitenoise. Zbyt mała ilość wolnej pamięci RAM.
write	Nie dotyczy	ENOMEM	Błąd alokacji pamięci w Whitenoise. Zbyt mała ilość wolnej pamięci RAM.
ioctl	WN_IOCTL_SET_LOCAL_ADDR WN_IOCTL_SET_REMOTE_ADDR WN_IOCTL_SET_CHANNELS WN_IOCTL_GET_CHANNELS WN_IOCTL_STATUS WN_IOCTL_OUTQUEUE_SIZE WN_IOCTL_INQUEUE_SIZE WN_IOCTL_WAIT	Brak	Funkcje te nie mogą zwrócić błędu. Wywołanie zawsze zwraca wartość 0.
	WN_IOCTL_LISTEN	EINVAL	Próbowano wywołać funkcję bez wcześniejszego ustawienia adresów źródłowego i/lub docelowego.
ioctl	WN_IOCTL_CONNECT	EINVAL	Próbowano wywołać funkcję bez wcześniejszego ustawienia adresów źródłowego i/lub docelowego.
		EBUSY	Połączenie nie znajduje się w stanie IDLE, najprawdopodobniej połączenie już zostało nawiązane lub jest w stanie LISTEN.
		EAGAIN	Błąd przy wysyłaniu pakietu. Spróbuj ponownie.
	WN_IOCTL_CLOSE	EBUSY	Trwa jeszcze transmisja bloku lub w kolejce znajdują się niewysłane dane.
		EAGAIN	Błąd przy wysyłaniu pakietu. Spróbuj ponownie.

6 Testy poprawności oraz efektywności działania protokołu

6.1 Środowisko testowe

Środowisko testowe do testów protokołu Whitenoise złożone zostało z trzech komputerów klasy PC:

- **Komputer 1** – z procesorem AMD Athlon 1,4 GHZ i 256 MB RAM. Zainstalowany system operacyjny: Linux Slackware 11.0 z kernelem 2.6.24.
- **Komputer 2** – z procesorem Intel Centrino 1.6 GHZ i 1 GB RAM. Zainstalowany system operacyjny: Linux Debian 4.0 z kernelem 2.6.24.
- **Komputer 3** – z procesorem Intel Centrino Duo 1.5 GHZ i 2 GB RAM. Zainstalowany system operacyjny: Linux Debian 4.0 z kernelem 2.6.24.

Komputery 1 i 2 zostały połączone ze sobą w sieć komputerową Ethernet 100 Mbps, Komputer 3 został połączony z Komputerami 1 i 2 za pomocą publicznej sieci IP o przepustowości 1 Mbps. Dodatkowe szczegóły techniczne dotyczące konfiguracji i użytego oprogramowania zawarte są w opisach poszczególnych testów w których były używane.

Do transmisji danych poprzez ukryty kanał informacyjny użyto aplikacji przestrzeni użytkownika o kodzie źródłowym przedstawionych w Załączniku 1 oraz Załączniku 2. Do testów z użyciem transmisji dwukierunkowej nieznacznie zmodyfikowano te aplikacje.

6.2 Testy

6.2.1 I test poprawności protokołu

Pierwszy test poprawności protokołu polegał na przesłaniu pomiędzy Komputerem 1, a Komputerem 2 100 MB pliku z losową zawartością (wygenerowanego linuksowym urządzeniem */dev/urandom*). Zarówno po stronie nadawcy, jak i odbiorcy z pliku testowego została wygenerowana suma kontrolna MD5 aby sprawdzić, czy plik został przesłany w postaci niezmienionej. Każdy z poniższych testów przeprowadzany był pięciokrotnie.

W celu wygenerowania niezbędnego ruchu sieciowego pomiędzy oboma komputerami transmitowane były duże pliki za pomocą polecenia *netcat* tak długo, aż testowy plik nie został przetransmitowany ukrytym kanałem w całości. Ze względu na charakterystykę ukrytych kanałów opartych o metodę TCP ISN przy testowaniu jej poprawności ruch sieciowy generowany był za pomocą autorskiej aplikacji nawiązującej i zamykającej z dużą szybkością połączenia TCP. W przypadku testowania kanału opartego jednocześnie o metody IP ID i TCP ISN ruch sieciowy generowany był równocześnie poprzez wyżej wymienione aplikacje.

Test kanału opartego o metodę IP ID
Suma kontrolna pliku źródłowego: 7c4222d2fbf789c74234e79e5c01632f
Suma kontrolna pliku docelowego: 7c4222d2fbf789c74234e79e5c01632f
Wynik testu: plik został przesłany poprawnie, protokół i metoda IP ID są skuteczne.

Test kanału opartego o metodę TCP ISN
Suma kontrolna pliku źródłowego: 7c4222d2fbf789c74234e79e5c01632f
Suma kontrolna pliku docelowego: 7c4222d2fbf789c74234e79e5c01632f
Wynik testu: plik został przesłany poprawnie, metoda osadzania danych w numerach ISN działa bez zarzutu.

Test kanału opartego o metody TCP ISN i IP ID:
Suma kontrolna pliku źródłowego: 7c4222d2fbf789c74234e79e5c01632f
Suma kontrolna pliku docelowego: 7c4222d2fbf789c74234e79e5c01632f
Wynik testu: plik został przesłany poprawnie, obie metody osadzania danych – używane jednocześnie – pracują bez zarzutu.

6.2.2 II Test poprawności protokołu

Drugi test poprawności protokołu obejmował przesłanie 20 MB pliku z losową zawartością (wygenerowanego linuksowym urządzeniem /dev/urandom) pomiędzy Komputerem 1 a Komputerem 3. Zarówno po stronie nadawcy, jak i odbiorcy z pliku testowego została wygenerowana suma kontrolna MD5 aby sprawdzić, czy plik został przesłany w postaci niezmienionej. Każdy z poniższych testów przeprowadzany był pięciokrotnie.

Test ten miał za zadanie sprawdzić mechanizmy detekcji oraz korekcji błędów w protokole. Dlatego też użyto do wymiany danych sieć publiczną IP, dodatkowo na Komputerze 3 za pomocą mechanizmów wbudowanych w jądro systemu Linux symulowano zakłócenia na łączu – 10 % wszystkich przekazywanych pakietów było gubionych. Niezbędny ruch sieciowy był generowany identycznie jak w teście opisanym w punkcie 6.2.1.

Test kanału opartego o metodę IP ID
Suma kontrolna pliku źródłowego: 8cc91d46d7f2a9a48d4f125ee75cc1c9
Suma kontrolna pliku docelowego: 8cc91d46d7f2a9a48d4f125ee75cc1c9
Wynik testu: plik został przesłany poprawnie, mechanizmy kontroli i korekcji błędów działają bez zarzutu.

Test kanału opartego o metodę TCP ISN
Suma kontrolna pliku źródłowego: 8cc91d46d7f2a9a48d4f125ee75cc1c9
Suma kontrolna pliku docelowego: 8cc91d46d7f2a9a48d4f125ee75cc1c9
Wynik testu: plik został przesłany poprawnie, mechanizmy kontroli i korekcji błędów działają bez zarzutu.

Test kanału opartego o metody TCP ISN i IP ID:
Suma kontrolna pliku źródłowego: 8cc91d46d7f2a9a48d4f125ee75cc1c9
Suma kontrolna pliku docelowego: 8cc91d46d7f2a9a48d4f125ee75cc1c9
Wynik testu: : plik został przesłany poprawnie, mechanizmy kontroli i korekcji błędów działają bez zarzutu.

6.2.3 III test poprawności protokołu

Trzeci test poprawności protokołu obejmował przesłanie dwóch 10 MB plików z losową zawartością (wygenerowanych linuksowym urządzeniem */dev/urandom*) jednocześnie w obie strony pomiędzy Komputerem 1 a Komputerem 3. Zarówno po stronie nadawcy, jak i odbiorcy z plików testowych została wygenerowana suma kontrolna MD5 aby sprawdzić, czy pliki zostały przesłane w postaci niezmienionej. Każdy z poniższych testów przeprowadzany był pięciokrotnie.

Test ten miał za zadanie sprawdzić poprawność działania oraz mechanizmy detekcji oraz korekcji błędów w protokole przy transmisjach dwukierunkowych. Dlatego też użyto do wymiany danych sieć publiczną IP, tym razem bez wprowadzania celowych zakłóceń. Niezbędny ruch sieciowy był generowany identycznie jak w teście opisanym w punkcie 6.2.1.

Przyjęto skróty: K1-K3 – plik transmitowany pomiędzy Komputerem 1, a Komputerem 3, K3-K1 – plik transmitowany pomiędzy Komputerem 3, a Komputerem 1.

Test kanału opartego o metodę IP ID
Suma kontrolna pliku źródłowego K1-K3: d779200b1cdc7d014c445dd2793f2896
Suma kontrolna pliku docelowego K1-K3: d779200b1cdc7d014c445dd2793f2896
Suma kontrolna pliku źródłowego K3-K1: 686185325f79247d19e7375f8839948d
Suma kontrolna pliku docelowego K3-K1: 686185325f79247d19e7375f8839948d

Wynik testu: pliki zostały przesłane poprawnie, mechanizm transmisji dwukierunkowej działa bez zarzutu.

Test kanału opartego o metodę TCP ISN
Suma kontrolna pliku źródłowego K1-K3: d779200b1cdc7d014c445dd2793f2896
Suma kontrolna pliku docelowego K1-K3: d779200b1cdc7d014c445dd2793f2896
Suma kontrolna pliku źródłowego K3-K1: 686185325f79247d19e7375f8839948d
Suma kontrolna pliku docelowego K3-K1: 686185325f79247d19e7375f8839948d

Wynik testu: pliki zostały przesłane poprawnie, mechanizm transmisji dwukierunkowej działa bez zarzutu.

Test kanału opartego o metody TCP ISN i IP ID:
Suma kontrolna pliku źródłowego K1-K3: d779200b1cdc7d014c445dd2793f2896
Suma kontrolna pliku docelowego K1-K3: d779200b1cdc7d014c445dd2793f2896
Suma kontrolna pliku źródłowego K3-K1: 686185325f79247d19e7375f8839948d
Suma kontrolna pliku docelowego K3-K1: 686185325f79247d19e7375f8839948d

Wynik testu: pliki zostały przesłane poprawnie, mechanizm transmisji dwukierunkowej działa bez zarzutu.

6.2.4 IV test poprawności protokołu

Czwarty test poprawności protokołu obejmował przesłanie dwóch 10 MB plików z losową zawartością (wygenerowanych linuksowym urządzeniem */dev/urandom*) jednocześnie pomiędzy Komputerem 1 a Komputerem 2 oraz Komputerem 3. Zarówno po stronie nadawcy, jak i odbiorców z plików testowych została wygenerowana suma kontrolna MD5 aby sprawdzić, czy pliki zostały przesłane w postaci niezmienionej. Każdy z poniższych testów przeprowadzany był pięciokrotnie.

Test ten miał za zadanie sprawdzić poprawność działania protokołu przy transmisjach równoległych. Niezbędny ruch sieciowy był generowany identycznie jak w teście opisanym w punkcie 6.2.1.

Przyjęto skróty: K1-K3 – plik transmitowany pomiędzy Komputerem 1, a Komputerem 3, K1-K2 – plik transmitowany pomiędzy Komputerem 1, a Komputerem 2, itd.

Test kanału opartego o metodę IP ID
Suma kontrolna pliku źródłowego K1-K2: d779200b1cdc7d014c445dd2793f2896
Suma kontrolna pliku docelowego K1-K2: d779200b1cdc7d014c445dd2793f2896
Suma kontrolna pliku źródłowego K1-K3: 686185325f79247d19e7375f8839948d
Suma kontrolna pliku docelowego K1-K3: 686185325f79247d19e7375f8839948d

Wynik testu: pliki zostały przesłane poprawnie, mechanizm transmisji równoległej działa bez zarzutu.

Test kanału opartego o metodę TCP ISN
Suma kontrolna pliku źródłowego K1-K2: d779200b1cdc7d014c445dd2793f2896
Suma kontrolna pliku docelowego K1-K2: d779200b1cdc7d014c445dd2793f2896
Suma kontrolna pliku źródłowego K1-K3: 686185325f79247d19e7375f8839948d
Suma kontrolna pliku docelowego K1-K3: 686185325f79247d19e7375f8839948d

Wynik testu: pliki zostały przesłane poprawnie, mechanizm transmisji równoległej działa bez zarzutu.

Test kanału opartego o metody TCP ISN i IP ID:
Suma kontrolna pliku źródłowego K1-K2: d779200b1cdc7d014c445dd2793f2896
Suma kontrolna pliku docelowego K1-K2: d779200b1cdc7d014c445dd2793f2896
Suma kontrolna pliku źródłowego K1-K3: 686185325f79247d19e7375f8839948d
Suma kontrolna pliku docelowego K1-K3: 686185325f79247d19e7375f8839948d

Wynik testu: pliki zostały przesłane poprawnie, mechanizm transmisji równoległej działa bez zarzutu.

6.2.6 Test przepustowości ukrytych kanałów

W przypadku przesyłania danych protokołem Whitenoise, szybkość ich transmisji nie zależy od szybkości transmisji jawnego ruchu sieciowego, lecz ilości pakietów w których można ukryć pakiety Whitenoise. O ile w przypadku ukrytego kanału opartego o pole IP ID ilość pakietów rzeczywiście rośnie wraz z szybkością transmisji (lecz ściśle zależy od średniego rozmiaru pakietu IP), o tyle już w przypadku metody TCP ISN już tak nie jest, a transfer ukrytych danych zależy od ilości nawiązywanych połączeń pomiędzy komputerami. W niniejszym badaniu wyznaczone zostaną zależności pomiędzy osiąganą szybkością ukrytej transmisji a ilością wymienianych pakietów IP i TCP.

Transmisje danych wykorzystane do badań były prowadzone pomiędzy Komputerem 1, a Komputerem 2. Pakiety IP i TCP służące do prowadzenia ukrytej transmisji generowane były sztucznie za pomocą narzędzia sieciowego *hping3*, z założeniem iż Komputer 2 odpowiada na każdy pakiet wysłany przez Komputer 1. Dzięki temu zawsze ilość pakietów wysłanych i odebranych z każdej maszyny jest taka sama, dając nam sumaryczną wartość pakietów wymienionych w danej jednostce czasu.
Szybkość wymieniania pakietów kontrolowana była za pomocą mechanizmów wbudowanych w narzędzie *hping3* i monitorowana za pomocą monitora sieciowego *iptraf*. Szybkość ukrytej transmisji mierzona była poprzez przesłanie pomiędzy komputerami pliku o rozmiarze 2 MB (i mierzenie czasu operacji) przy ustalonej wcześniej szybkości wymiany pakietów IP. Za każdym razem przeprowadzane były 3 próby, dając średnią.
W przypadku kanału opartego o metodę IP ID były to pakiety ICMP Echo Request oraz ICMP Echo Reply (Ping), a w przypadku kanału opartego o metodę TCP ISN były to pakiety TCP z ustawioną flagą SYN (skierowane na otwarty port, tak więc Komputer 2 prawidłowo odpowiadał pakietem SYN/ACK). W przypadku badania przepustowości kanału opartego o obie metody jednocześnie, używane były również pakiety TCP SYN.

Tab. 3, 4 oraz 5 oraz wykresy na Rys. 24 przedstawiają rezultaty pomiarów.

Tab. 3 Szybkość transmisji - metoda IP ID

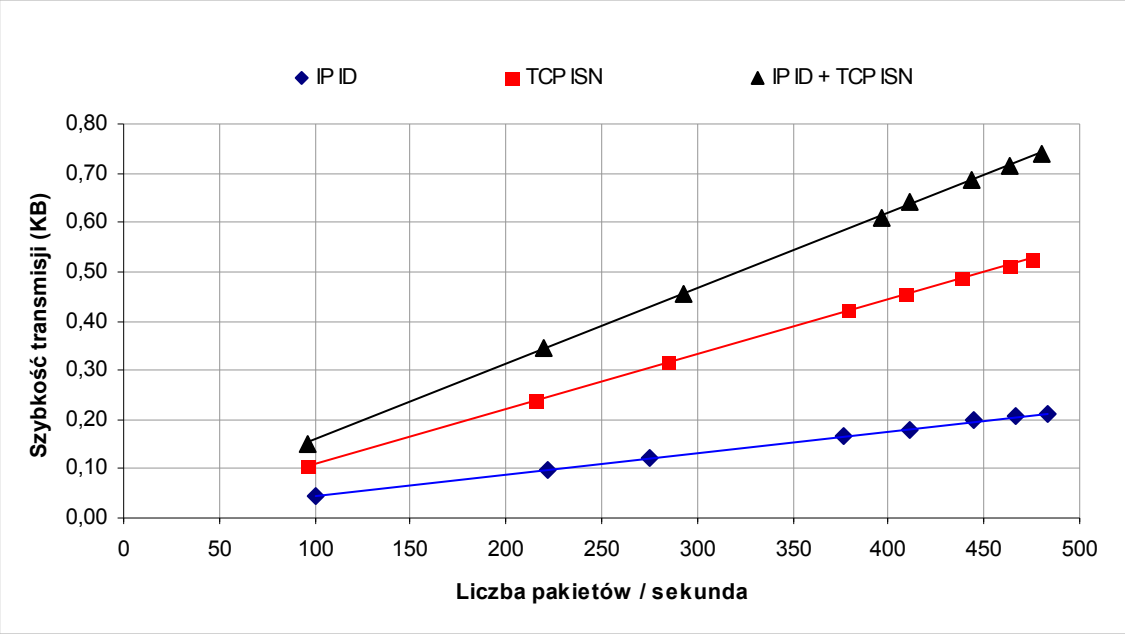
Szybkosc transmisji Whitenoise (KB/s)	Liczba wymienianych pakietów (pkt/s)
0,0427	100
0,0967	222
0,1200	275
0,1645	377
0,1789	411
0,1975	445
0,2058	467
0,2103	483

Tab. 4 Szybkość transmisji - metoda TCP ISN

Szybkośc transmisji Whitenoise (KB/s)	Liczba wymienianych pakietów (pkt/s)
0,1044	96
0,2380	215
0,3174	284
0,4208	379
0,4545	409
0,4878	438
0,5102	463
0,5250	475

Tab. 5 Szybkość transmisji - metody IP ID + TCP ISN

Szybkośc transmisji Whitenoise (KB/s)	Liczba wymienianych pakietów (pkt/s)
0,1515	96
0,3448	220
0,4545	293
0,6097	396
0,6410	411
0,6849	444
0,7142	463
0,7407	480



Rys. 24 Osiągana przepustowość transmisji Whitenoise

Na wykresie przedstawionym na Rys. 24 można zauważyć zależność prędkości transmisji w protokole Whitenoise od ilości wymienianych pakietów pomiędzy komputerami.

6.2.7 Badanie średniej ilości przesyłanych danych z użyciem naturalnego ruchu sieciowego

Celem niniejszego testu było sprawdzenie średniej ilości danych, którą można przesłać poprzez ukryty kanał informacyjny w trakcie wykonywania najpopularniejszych czynności w Internecie (takich jak np. przeglądanie stron informacyjnych czy sprawdzanie poczty e-mail).
Na potrzeby eksperymentu, na Komputerze 2 zainstalowano serwer proxy (z wyłączonym buforowaniem danych), a sam komputer podłączono do Internetu. Wszelkie próby łączenia się z Internetem (ruch HTTP) z Komputera 1 prowadziły przez serwer proxy Komputera 2. Pomiędzy Komputerem 1, a Komputerem 2 zestawiono ukryty kanał Whitenoise, monitorując ilość przesłanych danych. Dzięki takiemu rozwiązaniu, możliwa była symulacja normalnego ruchu sieciowego z zachowaniem warunków pozwalających na testy protokołu Whitenoise.

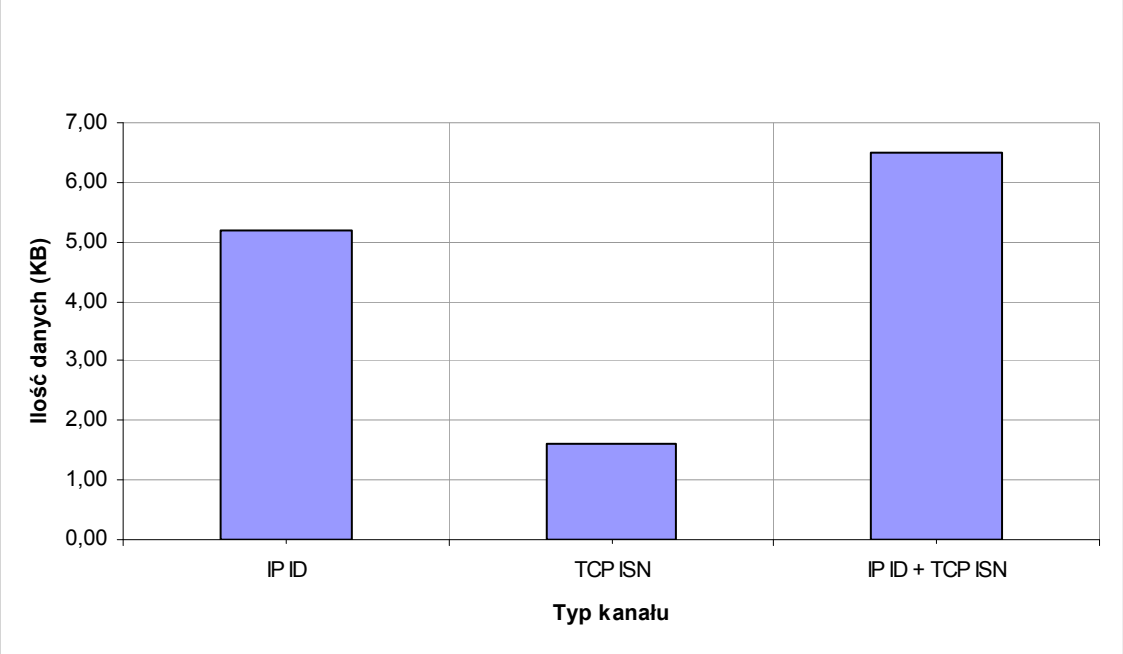
Scenariusz generowania ruchu sieciowego wyglądał następująco:

1. Otwarcie strony *http://gmail.com*, zalogowanie się do poczty e-mail oraz przeczytanie 5 wiadomości. Wylogowanie z serwisu
2. Otwarcie strony *http://www.onet.pl*, otwarcie po kolei pierwszych 5 artykułów z działu „Wiadomości” na stronie głównej serwisu.
3. Otwarcie strony *http://www.gazeta.pl*, otwarcie po kolei pierwszych 5 artykułów z działu „Sport” na stronie głównej serwisu

Test przeprowadzono z użyciem przeglądarki Mozilla Firefox w wersji 2.0.0.14 z wyłączoną obsługą pamięci cache. Każda próba (z podziałem na różne metody tworzenia kanałów Whitenoise) została przeprowadzona 4 razy w kilkugodzinnych odstępach czasowych (aby otwierane serwisy internetowe miały różną zawartość), a przedstawione wyniki obejmują osiągnięte wartości średnie. Wyniki badania, przedstawiające średnią ilość danych jaką przesłano poprzez kanał Whitenoise z użyciem wygenerowanego ruchu sieciowego znajdują się w Tab. 6 oraz na Rys. 25.

Tab. 6 Średnia ilość danych przesyłanych kanałem Whitenoise

Metoda	Ilośc danych (KB)
IP ID	5,2
TCP ISN	1,6
IP ID + TCP ISN	6,5



Rys. 25 Średnia ilość danych przesłana przez kanał Whitenoise

6.2.8 Test obciążenia procesora przez moduł Whitenoise

Celem niniejszego testu było zbadanie w jaki sposób działający moduł Whitenoise obciąża swoimi działaniami procesor komputera na którym pracuje. Jako, iż moduł składa się z dużej ilości procedur i uaktywniany jest on za każdym razem, gdy komputer odbierze pakiet IP istnieje podejrzenie, iż w sposób znaczny może on obciążać procesor i jądro systemu ciągłymi przełączeniami funkcji.

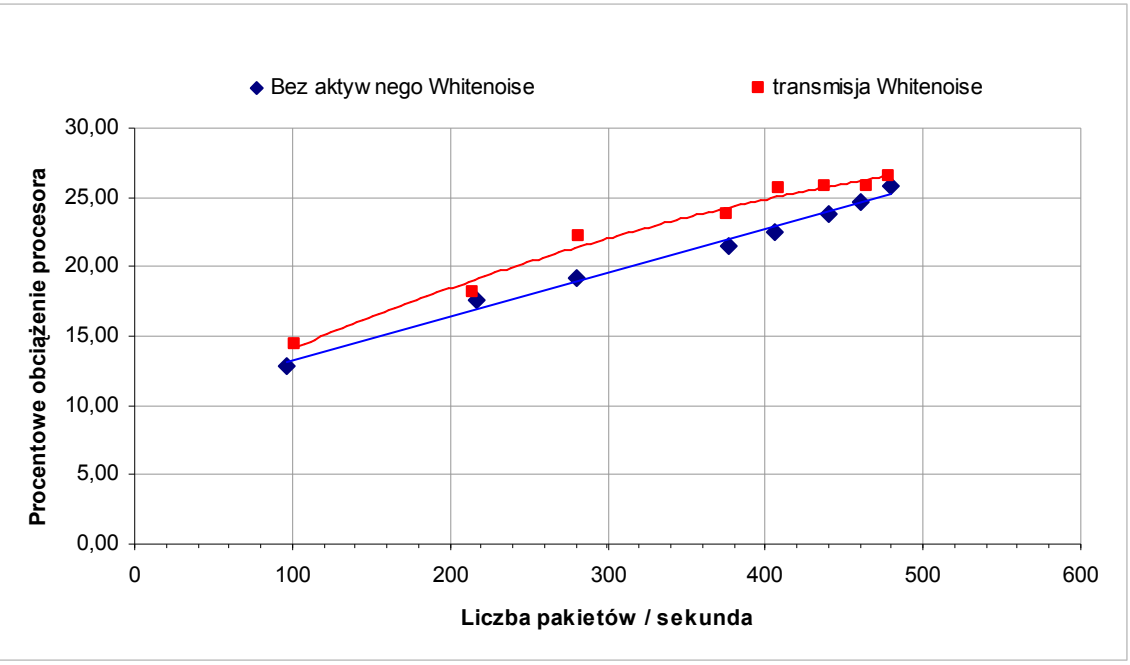
Badanie obciążenia procesora polegało na zmierzeniu obciążenia generowanego przez jądro systemu (bez załadowanego modułu Whitenoise) przy obsłudze pakietów przychodzących przez sieć z określoną częstotliwością. Następnie zaś zmierzenie obciążenia procesora generowanego przez jądro z uaktywnionym modułem Whitenoise, prowadzącym dwukierunkową transmisję kanałem opartym o metody IP ID i TCP ISN.
Obciążenie procesora mierzone było za pomocą pakietu *systtat*, dodając do siebie obciążenie generowane przez jądro, jak i procedury obsługujące przerwania i przerwania miękkie (*softirq*). Następnie była wyciągana średnia z całego czasu trwania pomiaru. Każdy pomiar prowadzony był przez 15 minut z zapisem aktualnego obciążenia prowadzonym co sekundę.
Badanie przeprowadzone zostało na Komputerze 1, drugą stroną transmisji był Komputer 2 – na nim też, za pomocą narzędzia *hping3*, były generowane pakiety TCP SYN będące nośnikiem transmisji.
Wyniki przeprowadzonego badania znajdują się w Tab. 7 oraz Tab. 8. Kolumna *liczba pakietów* określa łączną ilość pakietów wymienianych przez oba komputery (TCP SYN i TCP ACK generowane w odpowiedzi)

Tab. 7 Obciążenie procesora - bez aktywnego Whitenoise

Obciążenie systemu (%)	Liczba pakietów/s
12,8188	97
17,6641	217
19,2203	280
21,4526	377
22,4306	406
23,8272	440
24,6287	461
25,7780	480

Tab. 8 Obciążenie procesora - system z aktywną transmisją Whitenoise

Obciążenie systemu (%)	Liczba pakietów/s
14,4025	101
18,1079	215
22,1781	281
23,7988	375
25,7066	408
25,8080	438
25,8754	464
26,5709	478



Rys. 26 Obciążenie procesora

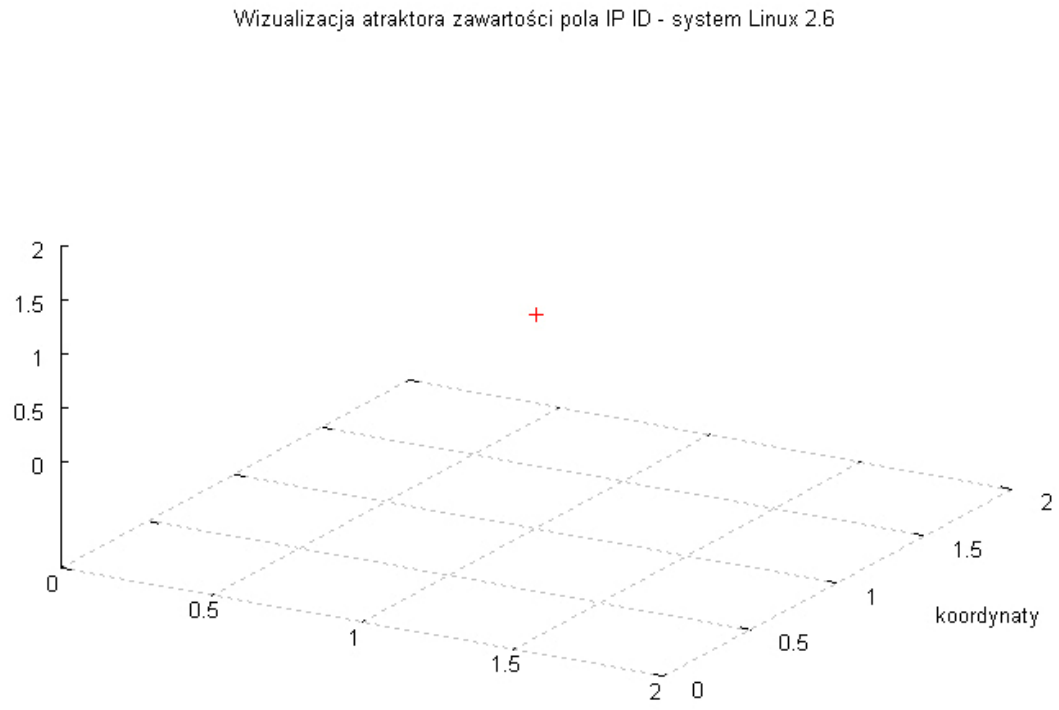
Jak widać na wynikach pomiarów przedstawionych Tab. 7 i Tab. 8 oraz wykresie na Rys. 26 pracujący moduł Whitenoise wprowadza zauważalne obciążenie procesora, lecz jest ono jedynie 2-3% większe od obciążenia generowanego przez jądro systemu bez aktywnego modułu.

6.2.10 Badanie wykrywalności ukrytych kanałów Whitenoise

Jako, że w przypadku Whitenoise ukryty kanał tworzony jest na bazie istniejącego i dopuszczonego przez politykę bezpieczeństwa ruchu sieciowego, unikane jest wychwycenie go za pomocą systemów detekcji włamań lub badania anomalii. Zaimplementowane metody ukrywania danych w pakietach (IP ID oraz TCP ISN) korzystają z pól nagłówków, które domyślnie zawierają liczby pseudolosowe, tak więc niemożliwe jest stwierdzenie, czy zawartość konkretnego pola w danej chwili została wygenerowana przez system operacyjny i nie ma żadnego ukrytego znaczenia czy może wręcz przeciwnie – jest nośnikiem ukrytej transmisji.
Jednak dogłębne badanie algorytmów generujących owe pseudolosowe liczby oraz zależności pomiędzy następującymi po sobie wartościami pozwalają na stworzenie charakterystyki danego generatora i porównania jej z wartościami obserwowanymi w sieci. Znając charakterystykę dla danego systemu operacyjnego, można zbadać czy wartości umieszczane w wysyłanych przez niego pakietach nie zostały gdzieś po drodze podmienione i tym samym mieć podstawę do podejrzenia istnienia ukrytego kanału informacyjnego. W niniejszej pracy do badania takich charakterystyk wykorzystane zostaną wizualizacje atraktorów generatorów liczb losowych, pozwalające na graficzne przedstawienie ich cech oraz złożonych zależności pomiędzy poszczególnymi wynikami ich pracy. Algorytm i metodologia badania zaczerpnięta została z pracy „Strange Attractors and TCP/IP Sequence

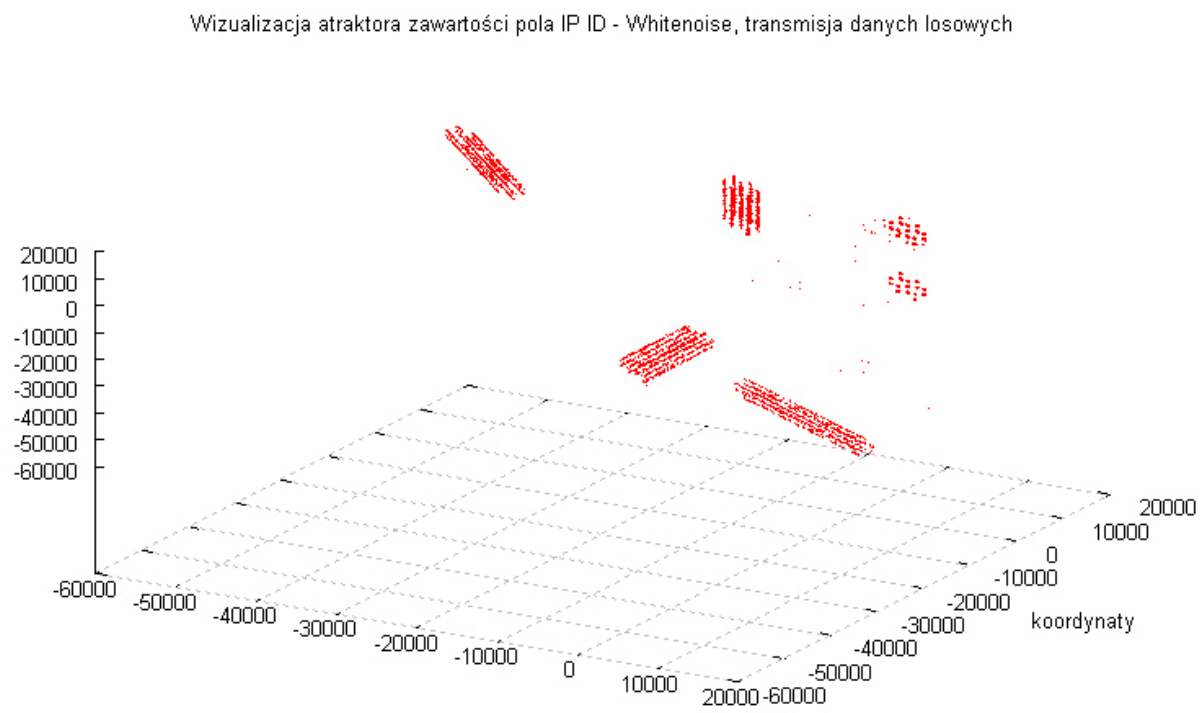
Number Analysis" [23] autorstwa Michała Zalewskiego. Przy badaniu numerów IP ID generowano testowy zbiór 15 000 wartości, zaś przy badaniu numerów TCP ISN generowano zbiór 50 000 wartości.

Jak zostało to zaznaczone we wcześniejszych rozdziałach, metoda ukrywania danych w polu IP ID powinna być używana jedynie do testów oraz zastosowań demonstracyjnych. Algorytm generowania wartości tego pola w systemie Linux (jak i najprawdopodobniej w wielu innych systemach) polega na kryptograficznym wyznaczeniu liczby początkowej (unikalnej dla danej pary adresów IP bądź połączenia TCP) i sekwencyjnym przypisywaniu każdemu pakietowi IP numeru większego o 1. Dla tak trywialnej charakterystyki wszelkie większe zaburzenia numeracji pakietów IP w ramach danego połączenia pomiędzy dwoma komputerami mogą wskazywać na celową modyfikację. Na Rys. 27 przedstawiono wizualizację atraktora dla algorytmu generującego kolejne wartości pola IP ID. Sprowadza się ona do jednego punktu w przestrzeni (o koordynatach 1,1,1).



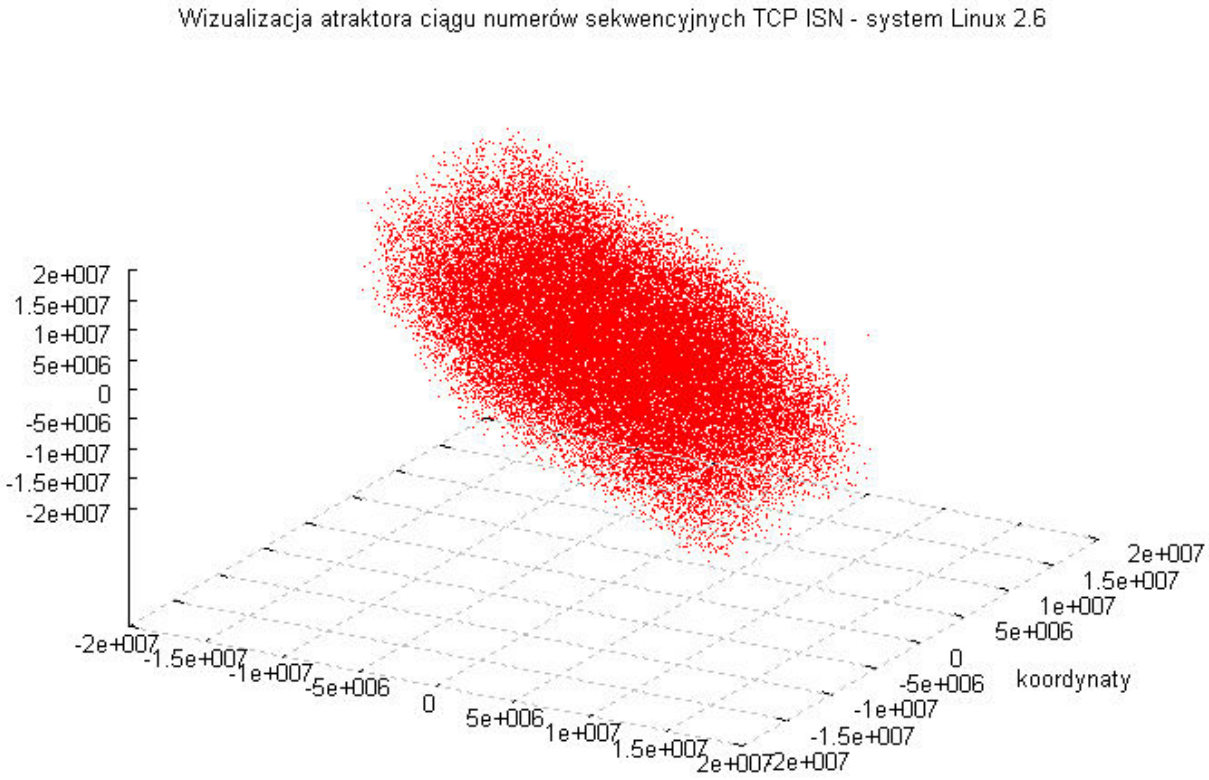
Rys. 27 Wizualizacja atraktora zawartości pola IP ID

W czasie prowadzenia transmisji Whitenoise, atraktor pola IP ID (przedstawiony na Rys. 28) zasadniczo zmienia swój wygląd - wyraźnie widać cztery wstęgi transmisji danych, a także kolisty zbiór punktów – najprawdopodobniej ten fragment charakterystyki generowany jest przez pakiety DATA_ACK.



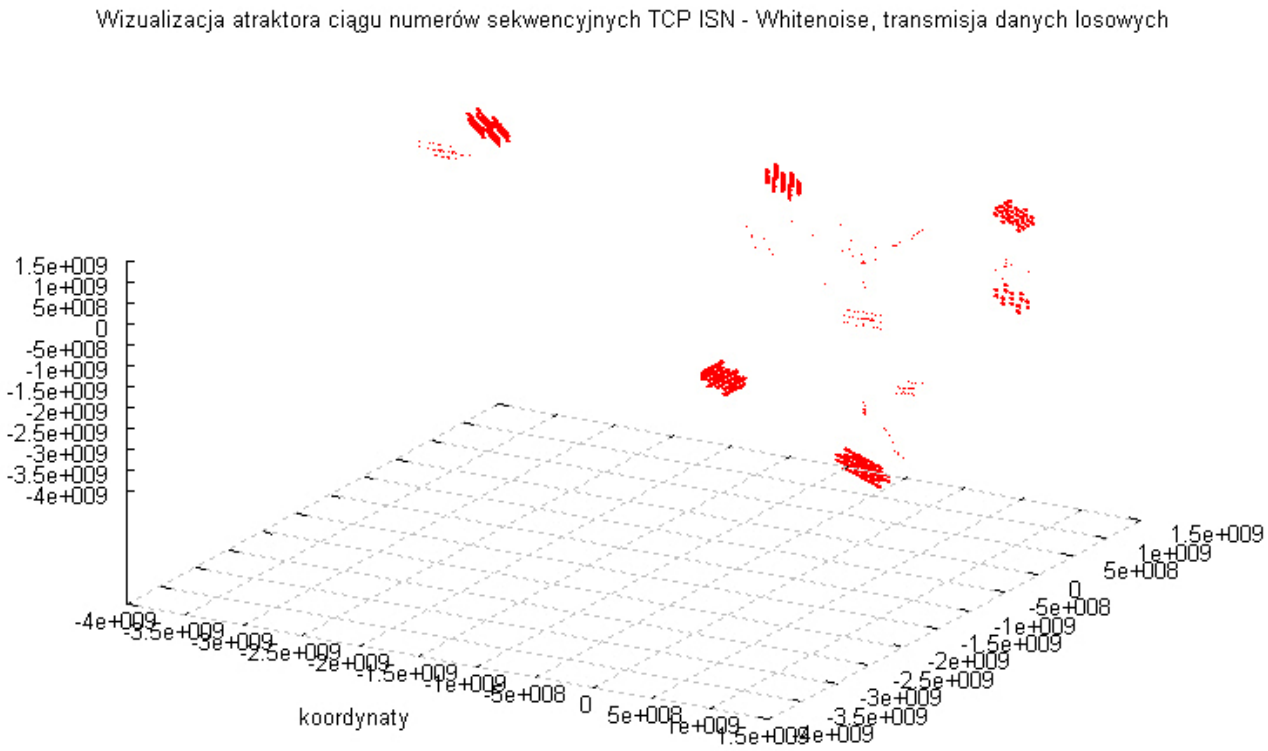
Rys. 28 Wizualizacja atraktora zawartości pola IP ID - Whitenoise, transmisja danych losowych

Nieco inaczej wygląda sprawa generowania numerów TCP ISN – ze względów bezpieczeństwa powinny być one w maksymalnym możliwym stopniu losowe. Jak widać na Rys. 29 w przypadku systemu Linux 2.6 tak jest w rzeczywistości – atraktor przyjmuje formę chmury, świadcząc o bardzo dobrej losowości sekwencji numerów ISN.

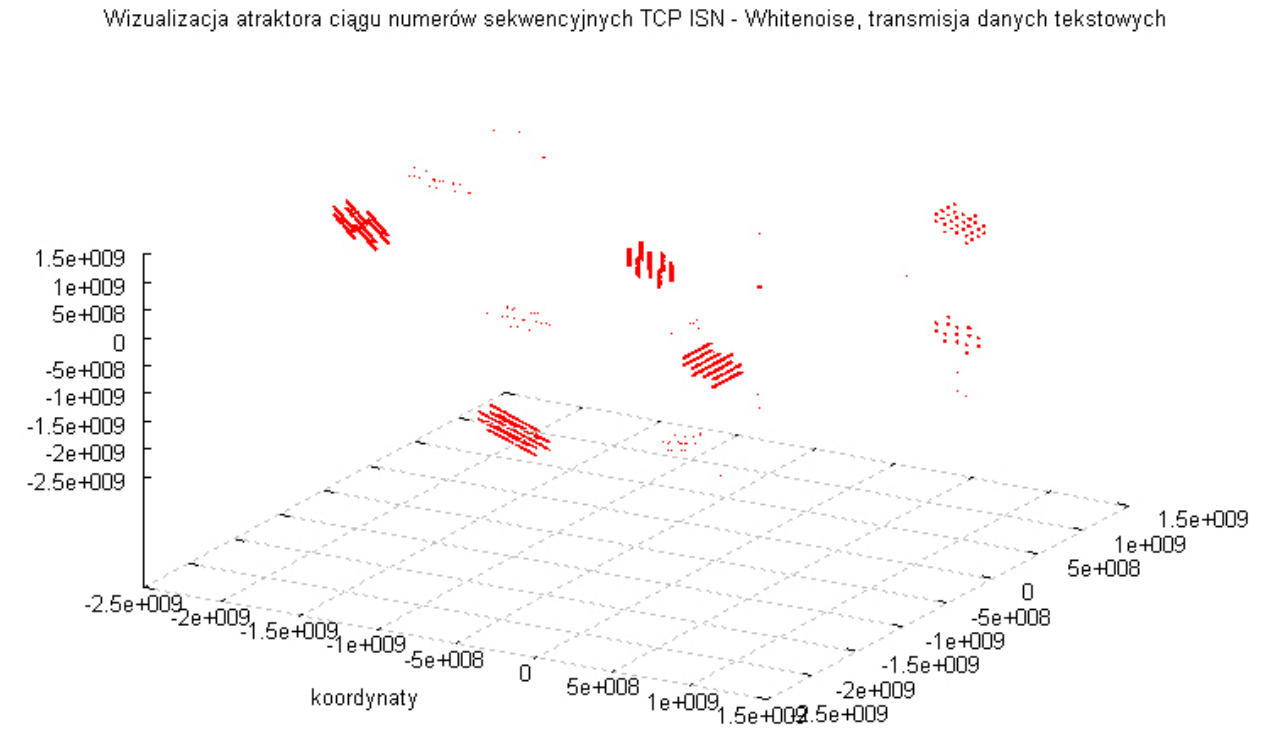


Rys. 29 Wizualizacja atraktora ciągu numerów sekwencyjnych TCP ISN

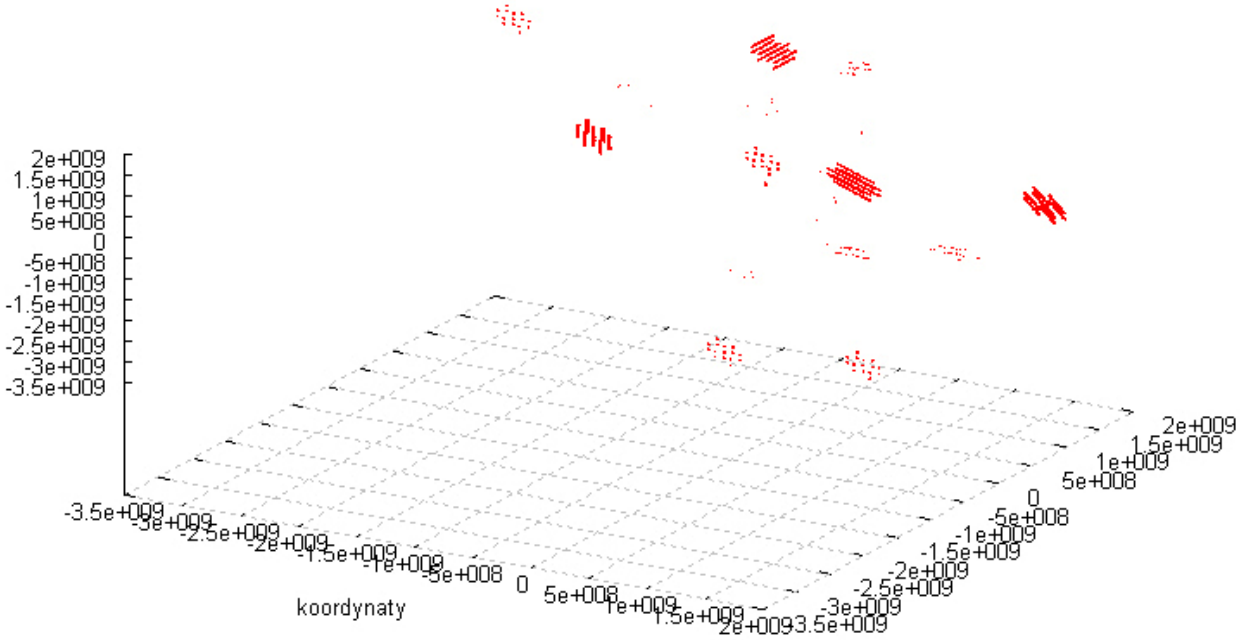
Jak nietrudno odgadnąć numery ISN zawierające transmisję Whitenoise takiej losowości nie będą miały – może być to podstawą do wykrycia modyfikacji numerów ISN i tym samym podejrzenia o istnienie ukrytego kanału informacyjnego. Udowadnia to atraktor zaprezentowany na Rys. 30 – transmisja ukrytym kanałem danych losowych wykazuje zupełnie inną charakterystykę – zmienną w małym stopniu od tego, czy przesyłamy dane losowe, czy np. tekst (patrz Rys. 31). Również zastosowanie szyfrowania pakietów Whitenoise (charakterystyka przedstawiona na Rys. 32) daje (co nie powinno dziwić) niewielkie rezultaty. Wynika z tego, iż kanały Whitenoise mogą być wykrywalne dla osób celowo szukających anomalii w charakterystykach losowości numerów ISN.



Rys. 30 Wizualizacja atraktora ciągu numerów sekwencyjnych TCP ISN - Whitenoise, transmisja danych losowych



Rys. 31 Wizualizacja atraktora ciągu numerów sekwencyjnych TCP ISN - Whitenoise, transmisja danych tekstowych



Rys. 32 Wizualizacja atraktora ciągu numerów sekwencyjnych TCP ISN - Whitenoise, zaszyfrowana transmisja danych tekstowych

Podsumowanie

Przeprowadzone w niniejszej pracy badania pokazują, iż zaprojektowany protokół transmisji, a także jego implementacja są w pełni funkcjonalne oraz niezawodne. Tym samym jest to dowód na to, iż dysponując możliwością przesyłania jedynie bardzo niewielkich porcji danych w niesprzyjających warunkach technicznych, możliwe jest stworzenie dwukierunkowego, ukrytego kanału informacyjnego. Test wykrywalności pokazuje jednak, iż dla osób analizujących rozkład statystyczny liczb losowych obecnych w TCP/IP, protokół Whitenoise może być łatwy do namierzenia. Zastosowany model szyfrowania danych okazał się mało skuteczny, dlatego konieczne jest skupienie się w dalszych pracach nad tym zagadnieniem na możliwościach generowania indywidualnego klucza dla każdego pakietu. Najlepszym w tym wypadku rozwiązaniem wydaje się zastosowanie algorytmu użytego w NUSHU [16], jednak rozszerzonego tak, aby klucz wyznaczany był również z zawartości pola TCP Timestamp. Otwarta architektura modułu implementującego protokół Whitenoise sprawia, iż jest on narzędziem dającym duże możliwości dalszych badań nad ukrytymi kanałami informacyjnymi. Obecny rozwój zagadnień dotyczących prywatności i bezpieczeństwa w Internecie pozwala mieć nadzieję, iż nie jest to ostatnia praca poświęcona tej technologii.

Bibliografia

1. Ahsan K., *Covert Channel Analysis and Data Hiding in TCP/IP*, http://gray-world.net/papers/ahsan02.pdf
2. Bertsekas D., Gallafer R, *Data Networks*, Prentice Hall, 1987
3. Cabuk S., Brodley C., Shields C., *IP Covert Timing Channels: An Initial Exploration*, http://www.cs.georgetown.edu/~clay/research/pubs/cabuk.ccs2004.pdf
4. Department of Defense USA, *Trusted Computer System Evaluation Criteria*, CSC-STD-00l-83, http://www.cs.cmu.edu/afs/cs/usr/bsy/security/CSC-STD-001-83.txt
5. Dokument *IETF 0791*, http://www.ietf.org/rfc/rfc0791.txt
6. Dokument *IETF 0793*, http://www.faqs.org/rfcs/rfc793.html
7. Dokument *IETF 1700*, http://www.faqs.org/rfcs/rfc1700.html
8. Giffin J, Greenstadt R., Litwack P., Tibbets R., *Covert Messaging Through TCP Timestamps*, Massachusetts Institute of Technology, http://www.springerlink.com/index/4D5JY0EWPLEA12D4.pdf
9. Hintz A., Covert Channels in TCP and IP Headers, 2003, http://www.defcon.org/images/defcon-10/dc-10-presentations/dc10-hintz-covert.ppt
10. ICMP Tunnel, http://www.detached.net/icmptunnel/
11. Kemmerer Richard A., *Shared resource matrix methodology: A practical approach to identifying covert channels*, ACM Transactions on Computer Systems, 1983, 1(3):256{277}
12. Lampson B.W, *A note on the confinement problem*, Proc. Of the Communications of the ACK, no 16:10, 1973, 613-615
13. Murdoch S.J, Lewis S., *Embedding Covert Channels into TCP/IP*, University of Cambridge, http://www.cl.cam.ac.uk/~sjm217/papers/ih05coverttcp.pdf
14. Ping Tunnel, http://www.cs.uit.no/~daniels/PingTunnel
15. Rowland C.H, *Covert channels in the TCP/IP protocol suite*, FirstMonday - Peer Reviewed Journal on the Internet, http://www.firstmonday.org/issues/issue2_5/rowland/
16. Rutkowska J., *Passive Covert Channels Implementation in Linux Kernel*, 2004, http://invisiblethings.org/papers/passive-covert-channels-linux.pdf
17. Shannon C.E., *A Mathematical Theory of Communication*, The Bell System Technical Journal, 1948, Vol. 27.
18. Standard *ISO/IEC 7498-1 : 1994*, http://www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_detail.htm?csnumber=20269
19. Stevens W. R., *UNIX. Programowanie usług sieciowych*, WNT Warszawa 2000 Wyd. I, 59
20. Qu H., Su P., Feng D., *A typical noisy covert channel in the IP protocol*, 38th Annual International Carnahan Conference on Security Technology, 189–192, http://ieeexplore.ieee.org/iel5/9638/30459/01405390.pdf?arnumber=1405390

21. Zander S, Armitage G., Branch P., *Covert Channels in the IP Time to Live Field*, Centre for Advanced Internet Architectures (CAIA), Swinburne University of Technology, http://www.ee.unimelb.edu.au/atnac2006/papers/52.pdf
22. Tumoian E., Anikeev M – *Detecting NUSHU Covert Channels Using Neural Networks*, http://www.ouah.org/neural_networks_vs_NUSHU.pdf
23. Zalewski M., *Strange Attractors and TCP/IP Sequence Number Analysis*, http://lcamtuf.coredump.cx/oldtcp/tcpseq.html

Spis rysunków

Spis tabel

7. Załączniki

7.1 Program klienta do transmisji plików z użyciem Whitenoise (w języku C)

```
#include <whitenoise_client.h>
#include <stdio.h>
#include <string.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <fcntl.h>
#include <errno.h>
#include <sys/stat.h>

#define WHITENOISE_FILE "/proc/whitenoise"

int main(int argc, char **argv){

     if(argc!=4){
          fprintf(stderr, "Usage: %s src_ip dst_ip filename\n",
argv[0]);
          exit(1);
     }

     int wfd, ffd, ret;
     unsigned int file_size, status, queue_size;
     struct stat st;
     struct wn_channels chan;
     in_addr_t local, remote;
     char buf[1024];
```

```
     if( (wfd=open(WHITENOISE_FILE, O_RDWR)) < 0){
          fprintf(stderr, "Error opening whitenoise (%s) file:
%s\n", WHITENOISE_FILE, strerror(errno));
          exit(1);
     }

     if( (ffd=open(argv[3], O_RDONLY)) < 0){
          fprintf(stderr, "Error opening file %s: %s\n",
argv[3], strerror(errno));
          exit(1);
     }


     local=inet_addr(argv[1]);
     remote=inet_addr(argv[2]);

     if(local==(in_addr_t)(-1) || remote==(in_addr_t)(-1)){
          fprintf(stderr, "Bad IP address format\n");
          exit(1);
     }


     if(fstat(ffd, &st)<0){
          fprintf(stderr, "Error getting file stat: %s\n",
strerror(errno));
          exit(1);
     }

     fprintf(stderr,"Size of the file to send is: %d bytes\n",
st.st_size);
     file_size=st.st_size;

     if(ioctl(wfd, WN_IOCTL_SET_LOCAL_ADDR, local)<0){
          fprintf(stderr, "Error in WN_IOCTL_SET_LOCAL_ADDR:
%s\n", strerror(errno));
          exit(1);
     }

     if(ioctl(wfd, WN_IOCTL_SET_REMOTE_ADDR, remote)<0){
          fprintf(stderr, "Error in WN_IOCTL_SET_REMOTE_ADDR:
%s\n", strerror(errno));
          exit(1);
     }

     bzero(&chan, sizeof(chan));

     chan.ip_id=1;
     chan.tcp_isn=0;
```

```
     ioctl(wfd, WN_IOCTL_SET_CHANNELS, &chan); //this can't
return error


     if(ioctl(wfd, WN_IOCTL_CONNECT)<0){
          fprintf(stderr, "Error in WN_IOCTL_CONNECT: %s\n",
strerror(errno));
          exit(1);
     }

     fprintf(stderr, "Connecting...\n");

     while(1){


     if(ioctl(wfd, WN_IOCTL_STATUS, &status)<0){
          fprintf(stderr, "Error in WN_IOCTL_STATUS: %s\n",
strerror(errno));
          exit(1);
     }

     if(status!=WN_STATUS_ESTABLISHED_ACTIVE){
          sleep(1);
          continue;
     }else{
          fprintf(stderr, "Connection established...sending
file.\n");
          break;
     }

     } //while(1)


     while( (ret=read(ffd, buf, 1024)) >0)
          write(wfd, buf, ret);

     if(ret<0){
          fprintf(stderr, "Error reading file: %s\n",
strerror(errno));
          exit(1);
     }

     fprintf(stderr, "All data queued. Waiting for
delivery...\n");

     while(1){

          ioctl(wfd, WN_IOCTL_OUTQUEUE_SIZE, &queue_size);

          if(queue_size>0){
```

```
                fprintf(stderr, "%d bytes left in queue.
Waiting.\n", queue_size);
                sleep(1);
                continue;
          }else{
                fprintf(stderr, "Queue empty. Closing
connection.\n");
                break;
          }
     } //while(1)

     if(ioctl(wfd, WN_IOCTL_CLOSE)<0){
          fprintf(stderr, "Error in WN_IOCTL_CLOSE: %s\n",
strerror(errno));
          exit(1);
     }

     ioctl(wfd, WN_IOCTL_WAIT);


exit(0);

}
```

7.2 Program serwera do transmisji plików z użyciem Whitenoise (w języku C)

```
#include <whitenoise_client.h>
#include <stdio.h>
#include <string.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <fcntl.h>
#include <errno.h>
#include <sys/stat.h>

#define WHITENOISE_FILE "/proc/whitenoise"

int main(int argc, char **argv){

     if(argc!=4){
          fprintf(stderr, "Usage: %s src_ip dst_ip filename\n",
argv[0]);
          exit(1);
     }

     int wfd, ffd, ret;
```

```
     unsigned int file_size, status;
     struct stat st;
     struct wn_channels chan;
     in_addr_t local, remote;
     char buf[1024];


     if( (wfd=open(WHITENOISE_FILE, O_RDWR)) < 0){
          fprintf(stderr, "Error opening whitenoise (%s) file:
%s\n", WHITENOISE_FILE, strerror(errno));
          exit(1);
     }

     if( (ffd=open(argv[3], O_WRONLY | O_TRUNC | O_CREAT)) <
0){
          fprintf(stderr, "Error opening/creating file %s:
%s\n", argv[3], strerror(errno));
          exit(1);
     }


     local=inet_addr(argv[1]);
     remote=inet_addr(argv[2]);

     if(local==(in_addr_t)(-1) || remote==(in_addr_t)(-1)){
          fprintf(stderr, "Bad IP address format\n");
          exit(1);
     }

     fprintf(stderr,"Saving data to file: %s\n", argv[3]);

     if(ioctl(wfd, WN_IOCTL_SET_LOCAL_ADDR, local)<0){
          fprintf(stderr, "Error in WN_IOCTL_SET_LOCAL_ADDR:
%s\n", strerror(errno));
          exit(1);
     }

     if(ioctl(wfd, WN_IOCTL_SET_REMOTE_ADDR, remote)<0){
          fprintf(stderr, "Error in WN_IOCTL_SET_REMOTE_ADDR:
%s\n", strerror(errno));
          exit(1);
     }

     bzero(&chan, sizeof(chan));

     chan.ip_id=1;
     chan.tcp_isn=0;

     ioctl(wfd, WN_IOCTL_SET_CHANNELS, &chan); //this can't
return error
```

```
     if(ioctl(wfd, WN_IOCTL_LISTEN)<0){
          fprintf(stderr, "Error in WN_IOCTL_LISTEN: %s\n",
strerror(errno));
          exit(1);
     }

     fprintf(stderr, "Waiting for connection\n");

     while(1){


     if(ioctl(wfd, WN_IOCTL_STATUS, &status)<0){
          fprintf(stderr, "Error in WN_IOCTL_STATUS: %s\n",
strerror(errno));
          exit(1);
     }

     if(status!=WN_STATUS_ESTABLISHED_PASSIVE){
          sleep(1);
          continue;
     }else{
          fprintf(stderr, "Connection established...receiving
file.\n");
          break;
     }

     } //while(1)

     while( (ret=read(wfd, buf, 1024)) >0){

          if(write(ffd, buf, ret)<ret){
               fprintf(stderr, "Error writing data to file:
%s\n", strerror(errno));
               exit(1);
          }

} //while (read...)

     if(ret<0){
          fprintf(stderr, "Error reading data from Whitenoise:
%s\n", strerror(errno));
          exit(1);
     }

     fprintf(stderr, "File received. Closing...\n");

     if(ioctl(wfd, WN_IOCTL_CLOSE)<0){
          fprintf(stderr, "Error in WN_IOCTL_CLOSE: %s\n",
strerror(errno));
          exit(1);
```

```
      }

           ioctl(wfd, WN_IOCTL_WAIT);

      close(wfd);
      close(ffd);

exit(0);

}
```