

# Informatyka II

## EE-DI, WEiI PRz

Instrukcja ta jest dostępna w formie źródłowej pod adresem [http://git.dms-serwis.com.pl/henrietta/inf2\\_eedi](http://git.dms-serwis.com.pl/henrietta/inf2_eedi).

Pod tym adresem będą pojawiać się również instrukcje do kolejnych laboratoriów.

## Laboratorium 1

### Laboratorium 1

*HTTP i DNS jako przykłady użycia TCP i UDP*

---

Z tego laboratorium przygotowujesz sprawozdanie. Przygotowujesz je na zajęciach, a przy ich zakończeniu wysyłasz na adres podany na końcu tej instrukcji. Będziesz musiał wkleić przynajmniej jeden obrazek, więc odpowiednio wybierz edytor.

Rzeczy oznaczone tak, jak poniżej, dotyczą tego, co masz zawrzeć w sprawozdaniu. Na przykład:

Zapisz swoje imię, nazwisko, adres e-mail, kierunek i rok studiów oraz grupę laboratoryjną i numer albumu.

Podaj również numer zajęć laboratoryjnych (nr 1) oraz numer zadania (to zadanie ma nr 9).

Mogą być to też pytania, na które w sprawozdaniu udzielisz odpowiedzi. Możesz pomagać sobie wyszukiwarką internetową, oraz zabrać głos w dyskusji, jeśli się jakaś wywiąże.

Teoria jest wpleciona w zadania. Po prostu czytaj, wykonuj polecenia i zadawaj sobie dużo pytań. Jeśli zupełnie utkniesz, a Google nie będzie Ci w stanie pomóc, zapytaj się prowadzącego. Chętnie wyjaśni Ci on również, dlaczego coś zrobiono *tak*, a nie *inaczej*.

## HTTP jako protokół tekstowy TCP

Jedną z rzeczy, którą początkującym informatykom trudno jest zrozumieć, jest różnica w protokołach TCP i UDP. Oba są protokołami warstwy transportowej OSI działającymi na podstawie protokołu IP (czyli do komunikacji należy mieć adres IP), jednak o nieco innych charakterystykach i zakresach użycia.

TCP, czyli *Transmission Control Protocol*, to **połączeniowy**, **niezawodny** i **strumieniowy** protokół.

- **połączeniowy** - zanim dojdzie do transmisji danych, należy zestawić *połączenie*. Obie maszyny muszą więc uzgodnić, że będą ze sobą rozmawiać. Nie można wysłać pakietu TCP tak po prostu (no, chyba że to pakiet *nawiąź połączenie*).
- **niezawodny** - jeśli programista powie “wyslij bajty A B C D”, to albo dojdą one kompletne, w całości, oraz tylko raz (słowem, tak jak zostały wysłane), albo nie dojdą wcale (awaria połączenia). Mimo, że pakiety IP mogą być po drodze gubione, a nawet duplikowane, TCP zapewnia że dane dojdą tak, jak zostały wysłane.
- **strumieniowy** - do TCP wysyłamy ciąg bajtów i otrzymujemy po drugiej stronie ciąg bajtów. Ze względu na jego *niezawodność*, te bajty mogą być dzielone, łączone, ogólnie mogą dziać się z nimi po drodze różne rzeczy. Jeśli wyślemy pakiet TCP o treści **Ala ma kota**, to niekoniecznie właśnie taki pakiet dojdzie do nadawcy. Mogą dojść do niego na przykład dwa pakiety - **Ala m**, oraz po jakimś czasie dopiero **a kota**. Nie mniej jednak, wszystko, czego wysłanie zostało zleczone, dojdzie w takiej kolejności. Jeśli TCP odbierze *późniejsze* dane przed *wcześniejszymi*, będzie “udawał” że ich nie ma, dopóki nie odbierze tych starszych. Nie nadaje się więc do zastosowań, gdzie *stare* pakiety po prostu nie są nam już potrzebne (np. wideokonferencje).

Z jednej strony więc rozpatrując TCP dostajemy pewne gwarancje, których sama warstwa sieciowa (w której działa tu IP), nam nigdy nie da, ale w pewnym sensie tracimy kontrolę nad swoimi pakietami. Pakiety mogą być gubione i przychodzić nie po kolei - TCP ułoży je oraz ponowi transmisję za nas. Pisząc programy korzystające z TCP **nie wolno zakładać** że dane dojdą podzielone tak, jak zostały wysłane. Po prostu są to ciągi bajtów. Odbiorca musi odesłać potwierdzenie odebrania danych, tak więc jest on mniej wydajny od UDP, pozwala jednak budować bardziej złożone protokoły nie martwiąc się o to, że pakiety nam zginą.

Z tego też powodu należało wymyślić sposób, jak zaznaczyć że jeden pakiet protokołu wyższej warstwy (np. takie zdanie) się kończy, a drugi zaczyna. Możliwe rozwiązania są trzy:

1. Będziemy na początku wysyłać jak długi jest pakiet, a dopiero potem pakiet.

2. Umówimy się, że jakiś znak (bajt) będzie rozdzielał pakiety.
3. Umówimy się, że każdy pakiet będzie miał identyczną, z góry znaną długość.

Możliwość 3 ogranicza nam pole manewru, więc odrzucimy ją. W praktyce wykorzystywane są pierwsze dwie opcje.

Protokoły binarne, czyli zrozumiałe przez maszyny, wykorzystują pierwszą możliwość, gdyż łatwo jest pisać programy korzystające z takich programów, jest to również bardziej wydajne. Niestety, protokoły binarne w czystej formie są bardzo trudne do zrozumienia dla człowieka bez użycia specjalnych narzędzi.

Na szczęście **HTTP** jest protokołem tekstowym. Oznacza to, że człowiek mógłby podejrzeć komunikację i bez większego problemu zrozumieć, o co chodzi. Nie zobaczy on *krzaków* (chyba że właśnie ściągamy za pomocą HTTP plik binarny), a czytelny tekst. Zastosowano w nim możliwość 2 - tym znakiem jest **znak końca linii**. - HTTP jest protokołem **synchronicznym**. Oznacza to, że przy nawiązanym połączeniu to klient wysyła zapytanie, a serwer odpowiada. Serwer sam z siebie nie może zacząć “mówić” (chyba że w *bardzo* szczególnych przypadkach). Po nawiązaniu połączenia (HTTP korzysta z TCP, więc musi najpierw je nawiązać) serwer czeka na żądanie klienta, a po jego otrzymaniu wysyła kod statusu, oraz odpowiedź. Żądanie takie zawiera *metodę* oraz *adres zasobu*.

Najbardziej znany kod statusu to HTTP 404. Pierwsza cyfra (setek) określa kategorię odpowiedzi. Poprawne odpowiedzi to kody 2xx, a informacje o przeniesieniu zasobu to 3xx. Błędy strony serwera (czyli nie z winy klienta) to 5xx.

Co to znaczy HTTP 404?

Wspominając o protokołach tekstowych raczej nie mówimy o pakietach, preferując raczej termin linia (czyli tekst do znaku końca linii).

Wadą protokołu tekstowego jest fakt, że maszyna musi się nieco bardziej wysilić, aby skorzystać z tego protokołu. Są one również wolniejsze. Zaletą jest ich zrozumiałość bez wyspecjalizowanego oprogramowania, co w początkach Internetu było bardzo ważne. Łatwiej też pisać *bardzo proste* programy, które mają z nich korzystać.

Z którego roku jest obecnie wykorzystywana wersja HTTP - HTTP/1.1?

Pobierz PuTTY. PuTTY jest terminalem, czyli uniwersalnym klientem protokołów tekstowych. Możemy więc nim obsłużyć wiele różnych protokołów. Zazwyczaj korzystają z niego administratorzy do logowania się na serwery, ale możemy nim również obsłużyć HTTP (choć nie jest to wygodne).

Zarówno TCP, jak i UDP do wysyłania danych wymagają podania (oprócz adresu IP) tzw. *portu*. Port to liczba od 0 do 65536.

Ilu bitów wymaga zapisanie jednego portu TCP/UDP?

Port służy do określenia *czego dokładnie* chcemy od docelowej maszyny. Może mieć ona bowiem wiele usług - np. WWW, poczta elektroniczna czy serwer plików. Port pozwala na określenie tej usługi. Znane protokoły mają również znane numery portów. Nadawaniem ich zajmuje się IANA - *Internet Assigned Numbers Authority* - jedna z organizacji zarządzających Internetem. Aby połączyć się za pomocą HTTP, musimy znać jego port.

Ustal, na którym porcie łączy się HTTP

Znajdź w sieci Internet, jak wygląda przykładowe żądanie i odpowiedź HTTP. Słowa kluczowe: **GET**, **HTTP/1.1**.

Zapisz tą przykładową parę, wraz z źródłem.

W pierwszej linii zawarto tzw. metodę HTTP, adres żadanego zasobu oraz wersję protokołu. Następnie występuje seria nagłówków żądania, czyli par *nazwa nagłówka* - *wartość*, które klient decyduje się podać serwerowi.

Czy przy użyciu PuTTY mógłbyś użyć protokołu HTTP/2.0? Dlaczego?

Gdy to zrobisz, uruchom PuTTY. Istotne są trzy pola:

- **Host Name** - nazwa docelowej strony. W twoim przypadku wpisz tu **www.python.org**
- **Port** - port, z którym będziemy się łączyć. Wpisz odpowiedni
- **Connection type** - w niektórych protokołach maszyna musi nam pomóc (np. bo są szyfrowane). HTTP takim nie jest, więc wybierz *Raw*

Jeśli połączenie zostanie poprawnie nawiązane, wyświetli się czarne okno. Wpisz kompletne zapytanie HTTP o Twoją stronę, sugerując się znalezionym przykładem.

Zapytaj o zasób główny (/) strony [www.python.org](http://www.python.org). Wzoruj się na znalezionym przykładzie. Wynikiem zapytania musi być kod 2xx lub 3xx.

Zapisz całość sesji. Abyś miał na to czas, przydatne może być użycie nagłówka Keep-Alive.

Najprawdopodobniej nie zadziała to bez użycia nagłówka Host. Pamiętaj, że adres strony, który wpisałeś, został zamieniony na adres IP przed połączeniem (a nie przesłany do serwera), spróbuj uzasadnić dlaczego.

Zaobserwuj odpowiedź. Składa się ona również z dwóch części - z listy nagłówków, oraz tzw. ciała. Tutaj odpowiedź była tekstem - kodem HTML. Był on zrozumiały dla człowieka. Typ zasobu, którym odpowiada serwer, znajduje się w nagłówku jego odpowiedzi jako *Content-Type*.

Kto przydziela typy MIME? Jakie typy MIME mogą mieć dokumenty Microsoft Word?

Spróbujmy otrzymać kod błędu. Kody błędów z winy klienta to 4xx, a z winy serwera to 5xx. Dużo łatwiej spowodować ten pierwszy, niż drugi. Nie każdy serwer HTTP zachowa się tak samo w obliczu błędów klienta - jeden może zignorować błędną metodę, a inny zgłosić HTTP 405. Kodom błędów (lub sukcesu) towarzyszy zazwyczaj krótki opis, np. *200 OK*, albo *404 Not Found*. Może być on inny niż ten ze specyfikacji.

Sprokuj serwer, z którym się łączysz, do zwrócenia błędu 404. Zapisz przebieg sesji.

Spróbujmy teraz pobrać zasób binarny. Ponownie za pomocą PuTTY połącz się ze stroną, ale tym razem pobierz zasób */favicon.ico*.

*/favicon.ico* to nazwa zasobu, który reprezentuje małą ikonkę stony. Zazwyczaj ikonka ta widoczna jest na pasku, obok nazwy karty. Jest to nazwa zwyczajowa, czyli jeśli przeglądarka stwierdzi istnienie takiego zasobu, to użyje go (chyba że twórca strony zażyczył sobie inaczej).

Pobierz *favicon.ico* z [www.python.org](http://www.python.org). Zanotuj typ MIME odpowiedzi, oraz całą sesję HTTP.

Jaki kod odpowiedzi otrzymałeś? Znajdź kogoś z innym kodem. Zanotuj jego numer zadania.

Serwer HTTP może więc przysyłać różne typy plików. Klient również może to zrobić. Musi on skorzystać jedynie z metody umożliwiającej przesłanie tzw. *ciała*. Wykorzystywane jego to do wysyłania formularzy, plików, itp.

Ustal jakie metody HTTP pozwalają na przesłanie ciała przez klienta.

Istnieją również inne protokoły tekstowe. Z nich również można, w pewnym stopniu, korzystać za pomocą PuTTY. Jak zobaczyliśmy nie jest to wygodne, ale pozwala nam dobrze zrozumieć protokół.

Przypomnij sobie definicję przeglądarki internetowej.  
Z jakiej właśnie skorzystałeś?

Wymień 3 rzeczy, które również można zrobić za pomocą  
PuTTY, oraz nazwę protokołu, który tą rzecz umożliwi.

Przeglądarki internetowe udostępniają zazwyczaj “podgląd” na to, co robią za pomocą HTTP w tzw. *narzędziach deweloperskich*. Można za pomocą ich zorientować się, jakie zapytania są realizowane, abyśmy mogli obejrzeć swoją stronę.

Uruchom narzędzia deweloperskie dowolnej przeglądarki. Wejdź na [www.python.org](http://www.python.org) i znajdź zapytanie, które było za to odpowiedzialne. Zrób mu zrzut ekranu i zamieść w sprawozdaniu.

Czy przeglądarka próbowała pytać się o `/favicon.ico`? Jeśli tak, to z jakim kodem skończyło się zapytanie? Zrób zrzut ekranu.

Skąd strony internetowe wiedzą, z jakiej przeglądarki korzystasz?

## DNS jako protokół binarny UDP

UDP (*User Datagram Protocol*) to drugi najczęściej wykorzystywany protokół warstwy transportowej korzystający z IP. Jest on protokołem *bezpoleźniowym* - czyli możemy do każdego wysłać pakiet UDP i nie musi się on go spodziewać. Pakiety UDP mogą jednak być:

- dostarczane nie po kolei
- duplikowane (wysyłasz jeden, dostajesz dwa)
- gubione

Podobnie jak TCP wykorzystuje on porty (ale port X w TCP to inny port niż X w UDP!). Jest to jednak w przeciwieństwie do TCP protokół *datagramowy*, czyli tutaj programista wysyła całe tzw. datagramy (czyli ciąg bajtów o ustalonej długości), a po drugiej stronie odbiera również te same pakiety (czyli taki sam ciąg bajtów). Nie są one dzielone, ani łączone, jak w TCP, ale mogą podlegać wcześniej wymienionym zjawiskom. Programista **musi o tym pamiętać**.

Jeśli programista wyśle większy pakiet UDP niż zalecany (zazwyczaj 512 bajtów), może on ulec fragmentacji. Zjawiska **fragmentacji UDP** należy unikać, bo wtedy możliwa jest sytuacja w której jeden fragment dotrze, a drugi nie, co skutkuje chaosem.

Ponieważ UDP nie wymaga pamiętania zestawionych połączeń, jest wydajniejszy od TCP. Dlatego też zastosowano go w protokole DNS, służącym zasadniczo do zamiany nazw domen postaci **wp.pl** na adresy postaci **212.77.98.9**. Dopiero taki adres umożliwia wysłanie czegoś (lub poproszenie o połączenie w TCP) do drugiej maszyny w sieci IP. Ma on jeszcze kilka innych zastosowań, o tym później.

Zauważmy, że pakiety UDP mogą być tracone lub duplikowane. Nie wpływa to na działanie DNS - zgubiony pakiet zawsze można odesłać, a zduplikowany stanu naszej wiedzy nie zmienia.

Na jakim porcie UDP działa DNS? A na jakim TCP?

Sam DNS (*Domain Name System*) przypisuje nazwie domeny listę par *typ rekordu + wartość*. Jedna domena może mieć kilka różnych rekordów. Zazwyczaj będzie zawierać adres serwera DNS, który jest odpowiedzialny za jej obsługę (czyli to właśnie w niego właściciel domeny wpisał jej dane), adres IP, adres serwera e-mail, oraz dodatkowe informacje. Serwer DNS może zwłaszcza wskazać na inny serwer DNS jako posiadający informację, której szukamy - nie musi być wszechwiedzący.

Jaka jest rola serwerów DNS root?

Wymień przynajmniej 3 organizacje, będące operatorami serwera root DNS.

Jest to protokół binarny, nie możemy więc wpisywać go ręcznie jako tekst. Musimy użyć specjalnego programu, tzw. *resolvera*. System Windows udostępnia wbudowanego klienta DNS o nazwie **nslookup**.

Otwórz linię poleceń Windows. Wywołaj narzędzie **\*\*nslookup\*\***. W jego linii poleceń wydaj polecenie **\*\*help\*\***. Przeczytaj uważnie i zrozum je.

Ustaw nslookup, aby udzielał wyczerpujących informacji o debugowaniu.

Ustal adres IP [www.python.org](http://www.python.org). Zanotuj odpowiedź nslookup.

Czy odpowiedź była autorytatywna? Co to oznacza? Jaki adres miał serwer DNS, który rozpatrzył Twoje zapytanie?

Serwery DNS mogą pracować w trybie rekursywnym, tj. jeśli nie wiedzą, to się dowiedzą, i dopiero odesłają zapytanie. Wymaga to od nich większej pracy, ale dzięki temu klient nie musi jej wykonać, a one na dodatek mogą umieścić domenę w pamięci podręcznej i odpowiedzieć w przyszłości "z pamięci". Każdy serwer

DNS zna adres do serwera obsługującego domenę poziom wyżej. Może również znać serwery DNS odpowiedzialne za domeny bardziej szczegółowe. Zobaczmy ten mechanizm w akcji.

Do tego celu skonfigurujemy **nslookup** tak, aby używał zapytań nierekursywnych oraz poszukiwania zaczął od jednego z serwerów root.

Wyłącz tryb rekursywny w nslookup. Ustaw jako domyślny serwer root. Zanonotuj polecenia, których użyłeś.

Wymyśl jakąś nazwę domeny .pl, z której ten komputer nie korzystał jeszcze od uruchomienia. Niech składa się ona przynajmniej z 3 części. Zapytaj o nią. Jeśli serwer nie udzieli odpowiedzi, a jedynie wskaże, jaki inny serwer obsługuje tę domenę, ustaw go jako domyślny w nslookup. Powtarzaj, aż nie uzyskasz adresu IP domeny.

Zanonotuj i skomentuj przebieg całej sesji. Z jakimi serwerami kontaktował się nslookup? Kto odpowiada za ich utrzymanie?

DNS może również przechowywać inne informacje na temat tej domeny. Służą do tego po prostu inne typy rekordów. Wykaz takich typów można znaleźć w sieci Internet.

Co oznacza typ rekordu PTR?

Jaki typ rekordu odpowiada za to, który serwer DNS odpowiada za daną domenę?

Wykonaj takie zapytanie (lub serię zapytań), aby otrzymać odpowiedź autorytatywną. Zanonotuj wydane polecenia i odpowiedź.

Dlaczego ta odpowiedź jest autorytatywna?

Rekord *MX* służy do ustalenia, gdzie nadać pocztę, zaadresowaną pod konkretny adres. Gdy wyślemy naszemu serwerowi poczty wychodzącej list do kogoś, musi on ustalić, pod jaki adres IP go dostarczyć. Mając adres `stefan@example.com` zapytamy się więc o rekord *MX* domeny `example.com`, uzyskując adres IP, a następnie serwer obsługujący nasze konto pocztowe spróbuje tam dostarczyć korespondencję.

Domeny można kupować. Służą do tego specjalne agencje, tzw. *registrarzy*. Po uiszczeniu opłaty rejestracyjnej, oraz dorocznej abonamentowej, domena jest *nasza*. Możemy wtedy decydować, co znajdzie się w wpisach DNS tej domeny. Na przykład, kupując `example.org` możemy zadecydować, co będzie w wpisach `inna.example.org`.

Wymień przynajmniej trzech polskich registrarów.

U którego najtaniej kupić na 3 lata domenę .pl?

Jeśli chcemy domenę od kogoś kupić, musimy poznać jego tożsamość. Zazwyczaj możemy dokonać tego za pomocą usługi WHOIS.

Do kogo należy domena `www.python.org`?

Podaj adres strony WHOIS z której korzystałeś, oraz całość odpowiedzi.

Niekiedy dane te nie są dostępne. Wtedy można skontaktować się z registrarzem, i poprosić o kontakt do tej osoby, lub przekazanie jej naszego kontaktu. Jeśli domena wykorzystywana jest niezgodnie z przeznaczeniem

Dlaczego przy domenach .pl do umieszczenia danych osoby fizycznej wymagana jest jej zgoda, a jeśli właścicielem jest firma, to już nie?

Jaka organizacja lub organizacje w Polsce rozpatrują spory związane z domenami?

Jeśli ciekawi Cię, czy sądy powszechne w Polsce respektują decyzje takich organizacji, to zapoznaj się pobieżnie z wyrokiem Sądu Apelacyjnego w Warszawie, sygn. akt. I ACa 1557/14.

## Zadania dodatkowe

Wykonaj je, jeśli masz czas. Nie są obowiązkowe, ale wykonanie ich nie zaszkodzi, a nawet pomoże. Odpowiedzi zamieść w sprawozdaniu.

1. Ostatnio wprowadzonym kodem błędu HTTP jest 451. Co on oznacza? Skąd taka wartość (i dlaczego)?
2. Czy oprócz TCP i UDP istnieją jakieś inne protokoły warstwy transportowej korzystające z IP? Jakież? Co robią?
3. Co to jest Sender Policy Framework? Z jakiego mechanizmu DNS korzysta?
4. Co to jest NASK? Jaką rolę spełnia w administracji domenami .pl?
5. HTTP w szyfrowanym wydaniu to HTTPS. Co zmieniono w protokole w stosunku do HTTP? Jak ułatwiono sobie tu życie za pomocą enkapsulacji i model ISO OSI?

## Wyślij sprawozdanie

To już prawie koniec zajęć. Zapisz swoje sprawozdanie.

Wyślij je na adres `sprawozdania@henrietta.com.pl`.  
W tytule umieść imię, nazwisko i numer zadania.  
Załącz sprawozdanie, lub wklej je w treść maila.

Gotowe!

## Do poczytania w wolnej chwili

- “The Law of Leaky Abstractions” - Joel Spolsky omawia TCP jako abstrakcję na IP, czyniąc ważne spostrzeżenia dla całej informatyki.
- “OSI: The Internet That Wasn’t” - czyli dlaczego model ISO OSI jest taki skomplikowany. Trochę historii Internetu.
- “Internet’s root servers take hit in DDoS attack”, czyli o tym, jak wyłączono 3 z 13 serwerów root DNS.
- “RFC-1149” - czyli gruchająca potęga enkapsulacji.

---

Copyright (c) 2017 Piotr Maślanka. Niektóre prawa zastrzeżone.

Kod źródłowy dostępny na [http://git.dms-serwis.com.pl/henrietta/inf2\\_eedi](http://git.dms-serwis.com.pl/henrietta/inf2_eedi).