



pasja-informatyki.pl

Sieci komputerowe

Protokoły warstwy transportowej

TCP i UDP

Damian Stelmach

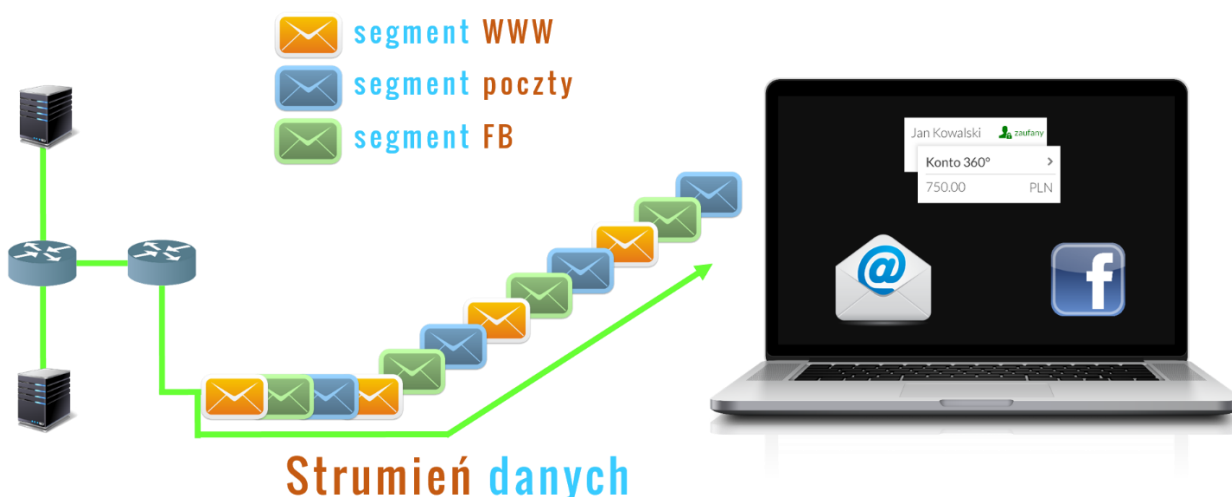
Spis treści

Zadania warstwy transportowej.....	3
Protokół TCP	7
Nagłówek TCP	7
Uzgadnianie trój-etapowe.....	8
Okno TCP	10
Protokół UDP	13
Polecenie NETSTAT	16

Warstwa transportowa czy też warstwa transportu, można stosować te nazwy naprzemiennie, to bardzo ważny element w procesie komunikacji. Do **najważniejszych zadań** tej warstwy zaliczyć należy:

- **nawiązanie i obsługa połączeń** (sesji) pomiędzy hostami,
- **śledzenie połączeń** pomiędzy hostami,
- **podział danych** na mniejsze fragmenty,
- **identyfikowanie** poszczególnych aplikacji,
- **kontrola przepływu danych**,
- **retransmisja** w przypadku utraty danych.

Śledzenie połączeń, czyli konwersacji pomiędzy hostami daje możliwość przesyłania i odbierania danych przez wiele aplikacji jednocześnie. Na jednym komputerze możemy przeglądać pocztę, korzystać z bankowości elektronicznej czy komunikować się ze znajomymi. Obecnie jest to dla nas naturalne, właściwie to trudno sobie wyobrazić sytuację, w której nie mielibyśmy takiej możliwości, ale warto przy tym pamiętać, że możliwe jest to między innymi dzięki warstwie transportowej. Na możliwość ciągłego korzystania z wielu usług jednocześnie składa się również segmentacja danych, czyli dzielenie ich na mniejsze fragmenty. Umożliwia to sprawniejszą komunikację, gdyż nie przesyła się jednocześnie dużej ilości danych. Gdyby nie segmentacja, to dane mogłaby odbierać jednocześnie tylko jedna aplikacja, pozostałe, z których korzystamy, musiałby czekać na swoją kolej. Widać to na grafice poniżej, segmenty przesyłane są na przemian, segment dla strony WWW, segment dla wiadomości e-mail, segment dla komunikatora i tak na przemian. Cały ten proces przesyłania segmentów wielu aplikacji na przemian nazywany jest **multipleksingiem**.



Kolejnym istotnym zadaniem czy funkcją warstwy transportu jest dostarczanie danych do właściwych aplikacji. Każda aplikacja posiada **swój identyfikator**, jednoznacznie ją określający. Identyfikatorem tym jest **numer portu aplikacji**.



port 80



port 5000



port 110



port 3074

Przypisywany jest on do segmentu lub datagramu w **procesie enkapsulacji** właśnie na poziomie warstwy transportu i gwarantuje on dostarczenie danych do konkretnej aplikacji.

**NUMER
PORTU**



**DODAWANY JEST
W PROCESIE
ENKAPSULACJI**

Podobnie jak w przypadku adresów IP, przydzielaniem numerów portów zajmuje się organizacja **IANA** (ang. Internet Assigned Numbers Authority), która to podzieliła numery portów na 3 grupy:

Nazwa grupy portów	Zakres numerów	Zastosowanie
Dobrze znane porty (ang. well knows)	0 - 1023	usługi i aplikacje serwera
Zarejestrowane porty (ang. registered)	1024 - 49151	usługi i aplikacje użytkownika
Dynamiczne porty (ang. dynamic)	49152 - 65535	losowo wybierane dla aplikacji klienta

Dobrze znane porty, czyli te od 0 do 1023 są zarejestrowane dla usług i konkretnych aplikacji serwerowych, przykładowo serwer WWW będzie domyślnie pracował na porcie 80, a serwer POP3 na 110. Zbiór aplikacji o dobrze znanych portach z uwzględnieniem protokołów warstwy transportowej przedstawiony jest poniżej.

Protokół warstwy aplikacji	Numer portu	Protokół warstwy transportowej
HTTP	80	TCP
HTTPS	443	TCP
POP3	110 (szyfrowany 995)	TCP
IMAP	143 (szyfrowany 993)	TCP
SMTP	25 (szyfrowany 465 lub 587)	TCP
FTP	21 (polecenia) i 20 (pliki)	TCP
FTPS	990	TCP
TELNET	23	TCP
SSH	22	TCP
DNS	53	TCP lub UDP
DHCP	67 i 68 (dla IPv6 546 i 547)	UDP
LDAP	389 (szyfrowany 639)	TCP lub UDP
SNMP	161	UDP

Druga grupa, czyli zarejestrowane porty wykorzystywane są przez aplikację zainstalowane na komputerze użytkownika. Jeśli przykładowo zainstalujemy na swoim komputerze aplikację będącą systemem zarządzania bazami danych MySQL, to będzie ona pracować na porcie 3306. Trzecia, ostatnia grupa czyli dynamiczne numery portów, z kolei są przydzielane losowo do aplikacji klienckich, np. kiedy klient wysyła żądanie udostępnienia strony WWW do serwera, to serwer przyjmuje to żądanie domyślnie na porcie 80, ale już odpowiedź, którą od serwera klient otrzymuje nie jest przesyłana na port 80, bo ten zarezerwowany jest dla procesów serwera WWW, ale na przydzielonym losowo numerze portów z puli portów dynamicznych. Działanie kilku aplikacji na tym samym numerze portu nie jest możliwe. Kiedy dana aplikacja pracuje na porcie np. 53 (DNS), to inna aplikacja już na tym porcie działać nie może, nie jest to możliwe.

Natomiast aplikacja czy usługa, jak wspomniany DNS, może korzystać zarówno z protokołu TCP, jak i UDP.

Jeśli wiemy już czym są porty aplikacji to wprowadźmy sobie kolejne pojęcie. Będzie nim **gniazdo** (ang. **Socket**), z pojęciem Socket'u spotkaliście się już przy okazji omawiania płyt główny i procesorów na zajęciach z urządzeń techniki komputerowej, w sieciach komputerowych również ono występuje. Gniazdo to kombinacja **adresu IP** i **numeru portu**.

212.167.21.201:80

Gniazdo **jednoznacznie identyfikuje** dany proces działający na urządzeniu i tak przykładowo, kiedy nasza przeglądarka będzie odwoływała się do serwera WWW o udostępnienie jakiejś strony internetowej, to zapytania do serwera zostanie przesłane do jego **gniazda**.



TCP to złożony, połączeniowy protokół, którego użycie ma gwarantować niezawodne dostarczenie danych oraz kontrolę przepływu. W procesie enkapsulacji, do nagłówka TCP dodawanych jest aż 20 bajtów danych sterujących, ale tego wymaga niezawodność TCP. Aplikacje korzystające z tego protokołu to przeglądarki internetowe, programy pocztowe czy programy do przesyłania plików. Wzór segmentu TCP widzicie poniżej. Liczby w nawiasach oznaczają ilość bitów, zarezerwowaną dla danego pola.

Nagłówek TCP

BIT (0)			BIT (15) BIT (16)		BIT (31)		
Port źródłowy (16)			Port docelowy (16)				
Numer sekwencyjny (32)							
Numer potwierdzenia (32)							
Długość nagłówka (4)		Zarezerwowane (6)		Bity kodu (flagi) (6)		Okno (16)	
Suma kontrolna (16)				Wskaźnik pilności (16)			
Opcje (0 lub 32 – jeśli istnieją)							
Dane warstwy aplikacji (dł. zmienna)							

Port źródłowy:

Port aplikacji, z której wysłano dane.

Port docelowy:

Port aplikacji, do której wysłano dane.

Numer sekwencyjny:

Numer ostatniego bajtu w segmencie.

Numer potwierdzenia:

Numer następnego bajtu oczekiwanego przez odbiorcę.

Długość:

Długość całego segmentu TCP.

Bitowy kod (flagi):

Informacje kontrolne dotyczące segmentu.

Okno:

Ilość danych jaka może zostać przesłana bez potwierdzenia.

Suma kontrolna:

Używana do sprawdzania poprawności przesłanych danych.

Wskaźnik pilności:

Używany tylko kiedy ustawiona jest flaga URG.

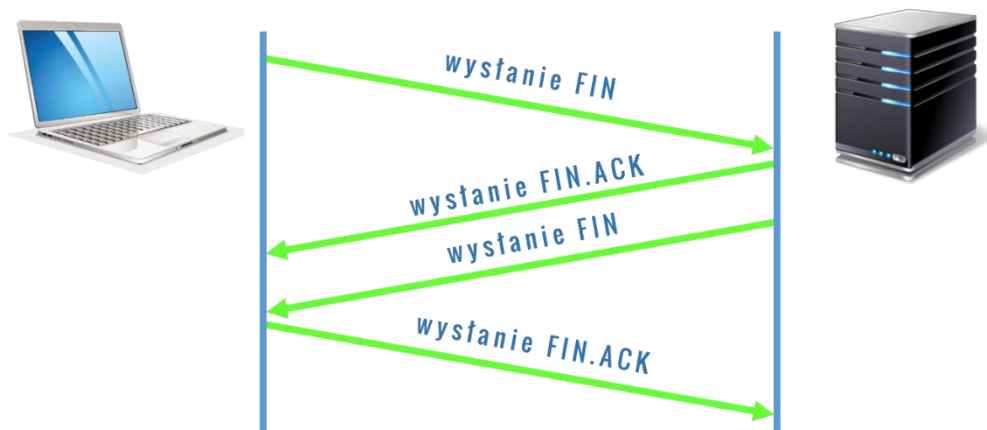
Uzgadnianie trój-etapowe

TCP jest **protokołem połączeniowym**, oznacza to, że zanim host źródłowy prześle jakiegokolwiek dane do hosta docelowego, to musi najpierw zostać ustanowione połączenie pomiędzy nimi. Połączenie to nosi nazwę **uzgadniania trój-etapowego** (ang. three-way handshake). **Host źródłowy**, czyli np. klient, wysyła segment zawierający flagę **SYN** (SYN to flaga synchronizacji numerów sekwencyjnych) W segmencie też zawarty jest losowy **numer sekwencyjny** klienta (zwany także numerem **ISN**, **SEQ=100**) służący do późniejszego scalania fragmentów danych. Odbierając ten segment **host docelowy**, czyli np. serwer jest informowany, że klient chce nawiązać z nim połączenie. Serwer w odpowiedzi wysyła segment z ustawioną flagą **SYN** i **ACK** (flaga ACK informuje klienta o tym, że serwer odebrał poprzedni segment, oraz że istnieje pole **numer potwierdzenia**), numerem sekwencyjnym odebrany od klienta **zwiększonym o 1 (ACK=101)** oraz swoim - losowym numerem sekwencyjnym (**SEQ=300**). Na koniec klient odsyła do serwera segment z ustawioną flagą **ACK** potwierdzającą odbiór poprzedniego komunikatu z **numerem sekwencyjnym** serwera **zwiększonym o 1 (SEQ=101, ACK=301)**. To kończy proces nawiązywania połączenia i pozawala na właściwą transmisję danych. Proces uzgadniania trój-etapowego widoczny jest poniżej.



UZGADNIANIE TRÓJ-ETAPOWE

Po tym jak wszystkie dane zostaną już przesłane, musi nastąpić zamknięcie sesji, wówczas klient, wysyła segment z flagą **FIN** do serwera, która to flaga informuje serwer o zamiarze zamknięcia sesji, ten odpowiada segmentem potwierdzającym z flagą **ACK**, iż taki segment odebrał. Następnie serwer również wysyła segment z flagą **FIN**, a klient odpowiada mu potwierdzającym segmentem z flagą **ACK**. To zamyka sesję TCP.



ZAMYKANIE SESJI

Bity kodu (flagi) stosowane do zarządzania sesjami TCP:

Flaga	Zastosowanie
URG	Informuje istnieniu pola wskaznik pilności w nagłówku (<i>urgent</i>)
ACK	Informuje istnieniu pola numer potwierdzenia w nagłówku (<i>acknowledgment</i>)
PSH	Wymuszenie przesłania pakietu (<i>push</i>)
RST	Ponowne zestawienie połączenia (<i>reset</i>)
SYN	Synchronizacja numerów sekwencyjnych
FIN	Koniec danych od nadawcy

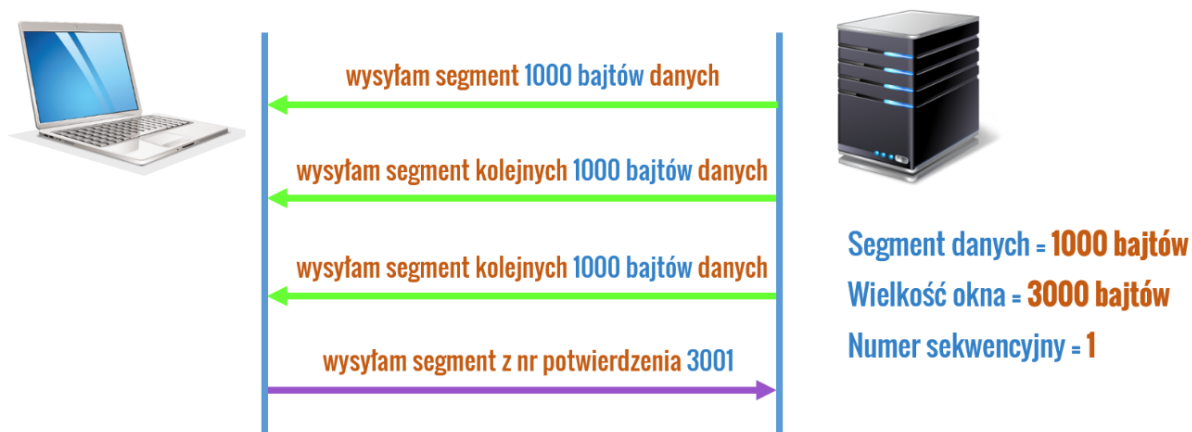
Okno TCP

Niezawodność dostarczania danych w sesji z wykorzystaniem protokołu TCP polega na wysyłaniu przez klienta **potwierzeń odbioru** wcześniej wysłanych danych. Zanim serwer prześle kolejną porcję danych do klienta, takie potwierdzenie odbioru musi odebrać. Czasami powoduje to opóźnienia w dostarczaniu segmentów ponieważ nie są one wysyłane w sposób ciągły. Uciążliwości te jednak są do przyjęcia, kiedy to wymagana jest niezawodność w komunikacji.

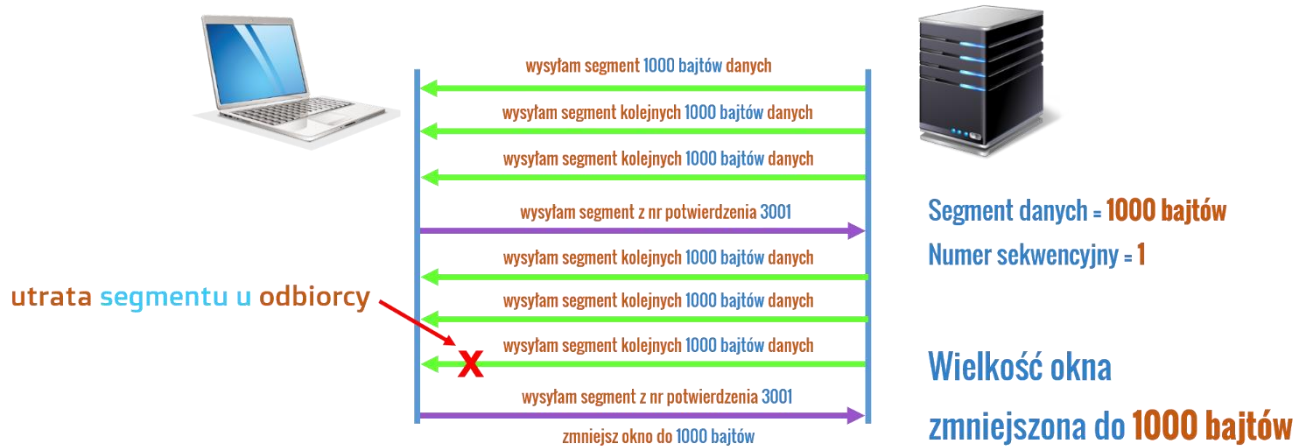
Przyjmijmy, że w ramach jednego segmentu wysyłanych jest **1000 bajtów** danych, wartość numeru sekwencyjnego to **1**. Kiedy klient odbierze 1 porcję danych, przesyła do serwera segment z **numerem potwierdzenia 1001**. Oznacza to mniej więcej taki komunikat: **odebrałem od ciebie 1000 bajtów, oczekuje na kolejne bajty, zaczynając od bajtu 1001**. Kiedy serwer wyśle kolejne 1000 bajtów, to numer potwierdza wysłany do niego jako informacja zwrotna będzie wynosił 2001, następny numer to już 3001, następny 4001 itd.



Oczywiście w rzeczywistości, kiedy to hosty musiałyby za każdym razem potwierdzać odebranie tak małej ilości danych spowodowałoby to spory zator na łączach, a przykładowo strony internetowe ładowałyby się bardzo długo. Dlatego też wysyła się więcej segmentów danych, które to potwierdzone są jedną informacją zwrotną. Ilość danych, jaka może być wysłana przez serwer zanim otrzyma on potwierdzenie od klienta nazywana jest **wielkością okna**, w tym przykładzie wynosi ona 3000 bajtów.



Wielkość ta określona jest w **nagłówku segmentu TCP** i oprócz tego, że określa, jaka ilość danych może zostać wysłana bez potwierdzenia, pozwala jeszcze na sterowanie przepływem danych pomiędzy urządzeniami. Jeśli na kliencie nastąpi **zator** w przyjmowaniu danych i dany segment zostanie utracony, urządzenie to może wysłać informacje do serwera, o zmniejszeniu wielkości tego okna, czyli ilości danych mogących zostać odebranych bez potwierdzenia, spowalnia to transmisję, ale zapobiega utracie segmentów. Po pewnym czasie wielkość okna przywracana jest to tej początkowej. Zmiana wielkości okna podczas transmisji nazywana jest **dynamicznym oknem** lub **oknem przesuwным**.



W przypadku drugiego protokołu tej warstwy, czyli **UDP**, jest zdecydowanie łatwiej, dlatego, że w tym protokole **nie zaimplementowano** mechanizmów gwarantujących niezawodność w dostarczeniu danych czy też kontroli przepływu.

Protokół UDP jest **prostym, bezpołączeniowym** protokołem, którego największą zaletą jest **niewielki narzut danych sterujących**, dodawanych w procesie enkapsulacji. UDP w datagramie dodaje tylko **8 bajtów** danych sterujących. Nagłówek datagramu UDP widzicie poniżej.

BIT (0)	BIT (15) BIT (16)	BIT (31)
Port źródłowy (16)	Port docelowy (16)	
Długość (16)	Suma kontrolna (16)	
Dane warstwy aplikacji (dł. zmienna)		

Port źródłowy:

Określa port aplikacji, z której wysłano dane.

Port docelowy:

Określa port aplikacji, do której wysłano dane.

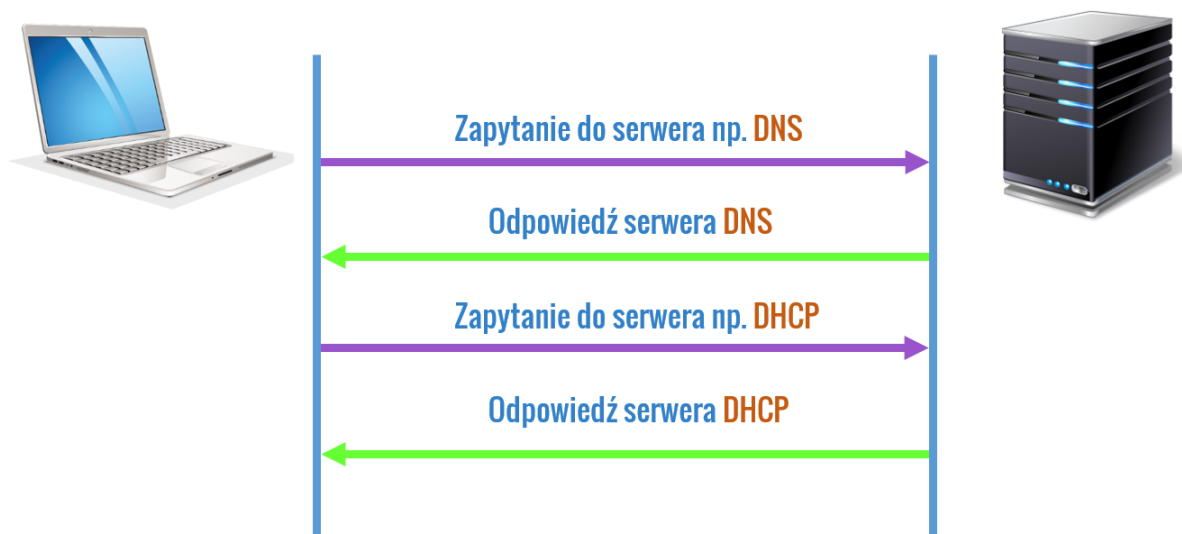
Długość:

16 - bitowe pole określające długość całego datagramu UDP

Suma kontrolna:

16 - bitowe pole służące do sprawdzania poprawności przesłanych danych.

Bezpołączeniowość protokołu UDP polega na tym, iż przed rozpoczęciem procesu komunikacji host źródłowy **nie wysyła** do hosta docelowego żadnych informacji, zestawiających to połączenie. Zasada jest taka, jeśli urządzenie źródłowe chce rozpocząć transmisję, chce wysłać dane po prostu to robi, bez wcześniejszego ustalenia.



Jeśli porównalibyśmy to do komunikacji ludzkiej to w przypadku protokołu TCP było by tak: **Hej, Tomek, skup się, bo zaraz będę do Ciebie coś mówił**, i dopiero po tym komunikacie rozpocząłbym właściwą rozmowę, oczywiście tylko wtedy kiedy Tomek odpowiedziałby mi komunikatem: **ok, zaczynam słuchać**. W przypadku UDP nie informuje Tomka, że zaraz zacznę przekazywać mu coś ważnego, po prostu zaczynam konwersację.

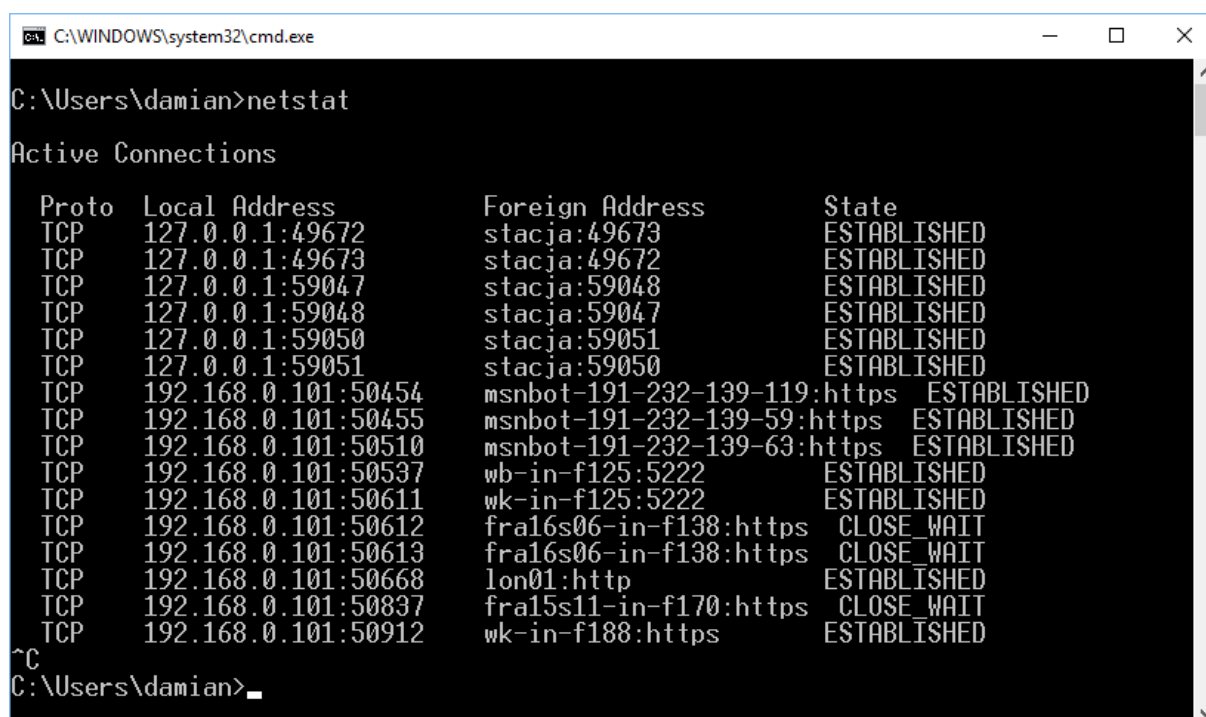
Aplikacje czy też usługi korzystające z tego protokołu to **DNS**, **DHCP**, **telefonia VoIP**, czy **streaming video**.



Dlaczego akurat te? No odpowiedź jest dość prosta, są to aplikacje, które nad niezawodność komunikacji, a właściwie powinienem powiedzieć nad konieczność odebrania całości danych, tak jak zostały one wysłane **cenia sobie szybkość**. Wyobraźmy sobie sytuację, kiedy oglądamy transmisję wideo czy gramy z kumplem, powiedzmy w CS'a. Trudno byłoby rywalizować w grze czy coś oglądać, kiedy to pakiety przychodziłyby z dużym opóźnieniem. Ktoś zapyta no ale skąd to opóźnienie miałoby się brać? No stąd chociażby, że segmenty TCP są sporo większe niż datagramy UDP, no i TCP wymaga potwierdzenia dostarczanych danych, dlatego też ich ilość

przesyłana przez sieć jest spora, większa niż w UDP. W przypadku aplikacji wykorzystujących ten właśnie protokół toleruje się to, że czasem jakiś pakiet może zostać utracony, bądź uszkodzony. W przypadku usługi DNS, jeśli datagram się zgubi to po prostu zostaje jeszcze raz wysłane zapytanie do serwera DNS, jeśli podczas telekonferencji jakiś datagram nie dotrze to też nie będzie tragedii, bo zawsze komunikat można powtórzyć. W przypadku aplikacji korzystających z TCP utrata czy zagubienie akceptowalne już nie jest. Datagramy odbierane są w takiej kolejności w jakiej zostały odebrane, a jeśli jest ich dużo, to za ich odpowiednie poskładanie odpowiada już konkretna aplikacja.

W jaki sposób w systemach Windows można podglądać zestawione połączenia naszego komputera z różnymi serwerami? Można do tego wykorzystać program **Wireshark**, dzięki któremu jesteśmy w stanie sprawdzić wszystko, co przechodzi przez naszą kartę sieciową, można również wykorzystać polecenie **NETSTAT** wykonywane w konsoli systemu Windows. Po jego wprowadzeniu możemy śledzić jakie mamy aktywne połączenia. Wynik działania polecenia pokazuje **rodzaj protokołu** warstwy transportowej wykorzystywany do połączenia, **gniazda mojego komputera**, czyli adres ip z numerami portów (widzicie tutaj, że te numery są przydzielane dynamicznie), **gniazda serwerów**, z którymi jesteśmy połączeni oraz **status tego połączenia**.



```
C:\WINDOWS\system32\cmd.exe
C:\Users\damian>netstat

Active Connections

Proto Local Address           Foreign Address         State
TCP    127.0.0.1:49672          stacja:49673           ESTABLISHED
TCP    127.0.0.1:49673          stacja:49672           ESTABLISHED
TCP    127.0.0.1:59047          stacja:59048           ESTABLISHED
TCP    127.0.0.1:59048          stacja:59047           ESTABLISHED
TCP    127.0.0.1:59050          stacja:59051           ESTABLISHED
TCP    127.0.0.1:59051          stacja:59050           ESTABLISHED
TCP    192.168.0.101:50454      msnbot-191-232-139-119:https ESTABLISHED
TCP    192.168.0.101:50455      msnbot-191-232-139-59:https ESTABLISHED
TCP    192.168.0.101:50510      msnbot-191-232-139-63:https ESTABLISHED
TCP    192.168.0.101:50537      wb-in-f125:5222         ESTABLISHED
TCP    192.168.0.101:50611      wk-in-f125:5222         ESTABLISHED
TCP    192.168.0.101:50612      fra16s06-in-f138:https  CLOSE_WAIT
TCP    192.168.0.101:50613      fra16s06-in-f138:https  CLOSE_WAIT
TCP    192.168.0.101:50668      lon01:http              ESTABLISHED
TCP    192.168.0.101:50837      fra15s11-in-f170:https  CLOSE_WAIT
TCP    192.168.0.101:50912      wk-in-f188:https        ESTABLISHED
^C
C:\Users\damian>
```

Program, wywoływany może być z różnymi parametrami, ich wykaz oraz opis wyświetli się po wpisaniu polecenia **netstat /help**.

Jak widzicie na grafice powyżej, jest sporo tych połączeń, a to dlatego, że po pierwsze korzystam z Windows 10, a ten system, jak powszechnie wiadomo niemal na okrągło przesyła coś na serwery Microsoft'u, ponadto mam ustawioną synchronizację z usługami w chmurze, no i jest antywirus, który również zestawia połączenia ze swoimi serwerami. Jak zatem zweryfikować z jakimi usługami łączy się nasz komputer? Wystarczy, że wywołamy polecenie **netstat -f** i skopiujemy nazwę domeny (PPM -> **oznacz** -> **zaznaczamy nazwę domeny** -> **CTRL+C** lub **PPM -> kopiuj**).


```
C:\WINDOWS\system32\cmd.exe

Proto Local Address          Foreign Address         State
TCP    127.0.0.1:49672          stacja:49673           ESTABLISHED
TCP    127.0.0.1:49673          stacja:49672           ESTABLISHED
TCP    127.0.0.1:59047          stacja:59048           ESTABLISHED
TCP    127.0.0.1:59048          stacja:59047           ESTABLISHED
TCP    127.0.0.1:59050          stacja:59051           ESTABLISHED
TCP    127.0.0.1:59051          stacja:59050           ESTABLISHED
TCP    192.168.0.101:50454      msnbot-191-232-139-119.search.msn.com:https ESTABLISHED
TCP    192.168.0.101:50455      msnbot-191-232-139-59.search.msn.com:https ESTABLISHED
TCP    192.168.0.101:50510      msnbot-191-232-139-63.search.msn.com:https ESTABLISHED
TCP    192.168.0.101:50537      wb-in-f125.1e100.net:5222 ESTABLISHED
TCP    192.168.0.101:50611      wk-in-f125.1e100.net:5222 ESTABLISHED
TCP    192.168.0.101:50612      fra16s06-in-f138.1e100.net:https CLOSE_WAIT
TCP    192.168.0.101:50613      fra16s06-in-f138.1e100.net:https CLOSE_WAIT
TCP    192.168.0.101:50668      lon01.ff.avast.com:http ESTABLISHED
TCP    192.168.0.101:50837      fra15s11-in-f170.1e100.net:https CLOSE_WAIT
TCP    192.168.0.101:50912      wk-in-f188.1e100.net:https ESTABLISHED
TCP    192.168.0.101:52559      r-149-58-45-5.ff.avast.com:http CLOSE_WAIT
TCP    192.168.0.101:53067      fra07s63-in-f13.1e100.net:https ESTABLISHED

C:\Users\damian>
```

Za pośrednictwem strony internetowej whois.domaintools.com i jej wyszukiwarki możemy sprawdzić właściciela domeny. Wystarczy, że wkleimy skopiowaną nazwę domeny. Jak widać właścicielem tej domeny jest Google.

1E100.net WHOIS, DNS, ...

whois.domaintools.com/1e100.net

HOME RESEARCH

PROFILE CONNECT MONITOR ACQUIRE SUPPORT WHOIS LOGIN Sign Up

Home > Whois Lookup > 1E100.net

Whois Record for 1E100.net

Find out more about Project Whois and DomainTools for Windows.

DOMAINTOOLS for Windows [Download Now](#)

Access domain ownership records from your desktop

Whois & Quick Stats

Email	abusecomplaints@markmonitor.com is associated with ~682,727 domains dns-admin@google.com is associated with ~17,784 domains
Registrant Org	Google Inc. is associated with ~18,413 other domains
Registrar	MARKMONITOR INC.
Registrar Status	clientDeleteProhibited, clientRenewProhibited, clientTransferProhibited, clientUpdateProhibited, serverDeleteProhibited, serverTransferProhibited, serverUpdateProhibited
Dates	Created on 2009-09-25 - Expires on 2019-09-25 - Updated on 2010-09-15
Name Server(s)	NS1.GOOGLE.COM (has 3,334 domains) NS2.GOOGLE.COM (has 3,334 domains) NS3.GOOGLE.COM (has 3,334 domains)

Tools

- Whois History
- Hosting History
- Monitor Domain Properties
- Reverse Whois Lookup
- Reverse Name Server Lookup
- Buy This Domain
- Visit Website
- Preview the Full Domain Report

No Screenshot Available

View Screenshot History

Available TLDs

General TLDs Country TLDs

The following domains are available through our platform