

Jak działa Internet – TCP i NAT

Witajcie w ostatnim artykule z serii „Jak działa Internet”! Zapoznaliśmy się w niej z różnymi zagadnieniami dotyczącymi Internetu – od najniższych warstw modelu OSI oraz adresowania IP i MAC po protokoły DNS i HTTP. Od czasu do czasu zdarza się jednak, że musimy przyjrzeć się warstwie „sklejającej” omówione dotąd mechanizmy. Dlatego w tym artykule poznamy podstawowe zagadnienia związane z protokołami TCP i UDP oraz zapoznamy się z procesem NAT.

I WPROWADZENIE DO TCP ORAZ UDP

Często spotykamy się z nazwą TCP/IP i chociaż nazwa zawiera protokół TCP i IP, to odnosi się do wszystkich omawianych i wspomnianych do tej pory protokołów (HTTP, DNS, ARP) [1]. W tym artykule skupimy się na protokołach TCP oraz UDP, które są częścią warstwy transportowej (czwartej warstwy modelu OSI). To one decydują o sposobie nawiązywania połączenia oraz walidacji poprawności otrzymanych danych.

I UDP

UDP (ang. *User Datagram Protocol*) jest protokołem znacznie prostszym niż TCP. Prostota ta jednak skutkuje ograniczonymi możliwościami. W ramach tego protokołu nie nawiązujemy połączenia ani nie otrzymujemy potwierdzenia dostarczenia pakietu. Korzystając z UDP, musimy liczyć się z tym, że niektóre pakiety mogą zostać utracone, a aplikacje muszą takie straty poprawnie obsłużyć. [5]

Zaletą UDP jest jednak jego „szybkość” w porównaniu do TCP – brak nawiązywania połączenia oraz brak mechanizmu potwierdzenia odbioru pakietów skutkuje mniejszą ilością przesyłanych danych. Dzięki temu jest on szeroko używany chociażby w strumieniowaniu wideo, grach online i czatów głosowych [6].

I TCP

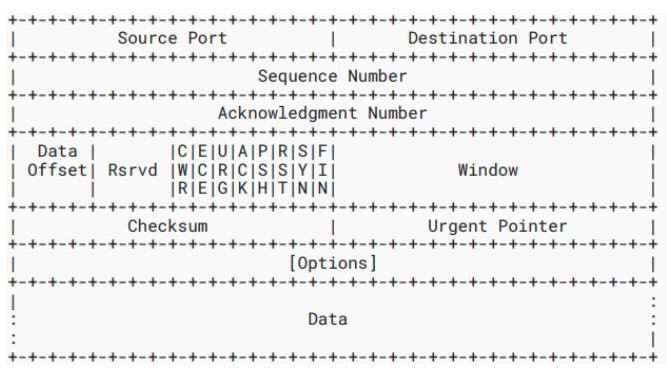
TCP jest niejako przeciwieństwem UDP zarówno co do narzutu danych, jak i funkcjonalności. Komunikacja w protokole TCP zaczyna się od nawiązania połączenia. Zarówno wysłane, jak i otrzymane pakiety są potwierdzane przez obie strony komunikacji. Wszystko to prowadzi do większej ilości wymienianych danych, ale za to mamy większe prawdopodobieństwo wykrycia błędów podczas ich przesyłu [7].

Ze względu na swoją charakterystykę protokół ten jest używany w większości aplikacji do komunikacji z serwisami oraz do transmisji plików. W tych przypadkach ważne jest, aby wiadomość dotarła kompletna bez błędów.

Zarówno protokół TCP, jak i UDP ma tzw. sumę kontrolną (ang. *checksum*). Suma ta zmniejsza prawdopodobieństwo zaakceptowania błędnego pakietu, ale mu nie zapobiega. Dużo lepszą integralność danych gwarantują dopiero protokoły wyższego poziomu jak np. TLS [16].

I SZCZEGÓŁY TCP

Tematyki tego protokołu dotknęliśmy już przy okazji omawiania protokołu HTTP. HTTP w większości przypadków używa właśnie TCP do transmisji danych, dlatego ważne jest dla nas jego głębsze zrozumienie. Na Rysunku 1 przedstawiono strukturę pakietu TCP podaną w rfc9293 (specyfikacji protokołu) [3].



Rysunek 1. Schemat pakietu według rfc9293

Widzimy wiele pól w pakiecie, a o najważniejszych z nich opowiemy w tym artykule [8]:

Port źródłowy i docelowy (source port, destination port) – to nic innego jak liczby, które pozwalają identyfikować komunikację między różnymi aplikacjami w ramach jednego hosta. Możemy to sobie wyobrazić na przykładzie przeglądarki internetowej z wieloma otwartymi zakładkami, na których jest wczytana ta sama strona internetowa. Ponieważ strona internetowa jest pewną aplikacją, to wszystkie pakiety TCP wysłane przez te zakładki będą miały ten sam port docelowy. Jednak różne zakładki mogą poruszać się po różnych podstronach witryny, żądając innych danych, dlatego każde takie połączenie do serwera będzie mieć inny port źródłowy. Dzięki temu serwer będzie mógł odróżnić połączenia od siebie i wysłać odpowiednie dane, używając odpowiedniego połączenia.

Numer sekwencyjny i numer potwierdzenia (sequence number i acknowledgement number) – liczby pozwalające stronom komunikacji na wzajemne informowanie się o wysłanych i odebranych danych, co umożliwia retransmisję zagubionych pakietów.

Data offset – informuje nas, gdzie w pakiecie znajdują się dane przeznaczone dla aplikacji. Wartość ta jest zmienna ze względu na pole Options, które ma zmienną długość.

Flagi (CWR, ECE, URG, ACK, RST, PSH, SYN, FIN) – bity określające przeznaczenie danego pakietu. Na przykład flaga SYN służy do nawiązania połączenia (synchronizacji), a PSH (push) do przesyłania danych.

Okno (ang. *Window*) – Informuje o wielkości bufora danych zarezerwowanego do komunikacji między klientem a serwerem. Jego użycie w praktyce zobaczymy w dalszej części artykułu.

Checksum – suma kontrolna pozwalająca na weryfikację poprawności przesłanych danych.

Urgent Pointer – obecnie jest głównie nieużywany. Nie będzie omawiany w tym artykule.

Options – dodatkowe opcje połączenia, które mogą być wspierane przez strony komunikacji. Pole może również zawierać dodatkowe dane przesyłane razem z pakietem, jak np. te omawiane w sekcji SACK.

Data – dane wysyłane do aplikacji w pakiecie TCP.

Spójrzmy na pakiety TCP w akcji. Wyślijmy żądanie HTTP do dowolnego serwisu (w tym przypadku <http://www.google.com>) i podejrzmy pakiety w aplikacji Wireshark.

Uruchommy więc Wireshark, a do wysłania żądania wykorzystamy narzędzie cURL, jak przedstawiono to w Listingu 1.

Listing 1. Wykorzystanie narzędzia cURL do wysłania żądania

```
curl http://www.google.com
```

W oknie Wiresharka pojawiają się nam kolejne pakiety TCP. Po ich odfiltrowaniu powinniśmy ujrzeć okno podobne do tego z Rysunku 2. Na jego podstawie przeanalizujemy przebieg komunikacji TCP.

Nawiązywanie połączenia TCP

Na początku komunikacji następuje proces nawiązywania połączenia. Często określa się go jako *three-way handshake*, ponieważ składa się z trzech wymienianych pakietów (klient → serwer, serwer → klient, klient → serwer). Warto wspomnieć, że jeden z tych pakietów wynika z połączenia pakietów SYN i ACK, które zobaczymy za chwilę. W związku z tym poprawne jest również nawiązywanie połączenia w czterech krokach.

Protokół TCP przewiduje także sytuację, w której obie strony jednocześnie inicjują nawiązanie połączenia. Chociaż jest to rzadkie zjawisko, warto o nim pamiętać, analizując mniej typowe scenariusze komunikacji TCP [9].

Korzystając z opcji Flow Graph (dostępnej w Statistics → Flow Graph), możemy lepiej zwizualizować przebieg komunikacji, co pokazuje Rysunek 3.

No.	Time	Source	Destination	Protocol	Length	Info
44	0.304333877	10.11.0.25	216.239.38.120	TCP	74	58250 → 80 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK_PERM TSval=1529650600 TSecr=0 WS=128
45	0.319159675	216.239.38.120	10.11.0.25	TCP	74	80 → 58250 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=1412 SACK_PERM TSval=2049743639 TSecr=1529650600 WS=256
46	0.319278570	10.11.0.25	216.239.38.120	TCP	66	58250 → 80 [ACK] Seq=1 Ack=1 Win=64256 Len=0 TSval=1529650615 TSecr=2049743639
47	0.319553379	10.11.0.25	216.239.38.120	HTTP	144	GET / HTTP/1.1
48	0.337382260	216.239.38.120	10.11.0.25	TCP	66	80 → 58250 [ACK] Seq=1 Ack=79 Win=268800 Len=0 TSval=2049743658 TSecr=1529650615
49	0.810539294	216.239.38.120	10.11.0.25	TCP	2866	80 → 58250 [PSH, ACK] Seq=1 Ack=79 Win=268800 Len=2800 TSval=2049744091 TSecr=1529650615 [TCP PDU reassembled in 6]
50	0.810539931	216.239.38.120	10.11.0.25	TCP	2866	80 → 58250 [PSH, ACK] Seq=2801 Ack=79 Win=268800 Len=2800 TSval=2049744091 TSecr=1529650615 [TCP PDU reassembled in 6]
51	0.810540214	216.239.38.120	10.11.0.25	TCP	2866	80 → 58250 [PSH, ACK] Seq=5601 Ack=79 Win=268800 Len=2800 TSval=2049744091 TSecr=1529650615 [TCP PDU reassembled in 6]
52	0.810540538	216.239.38.120	10.11.0.25	TCP	2866	80 → 58250 [PSH, ACK] Seq=8401 Ack=79 Win=268800 Len=2800 TSval=2049744091 TSecr=1529650615 [TCP PDU reassembled in 6]
53	0.810700524	10.11.0.25	216.239.38.120	TCP	66	58250 → 80 [ACK] Seq=79 Ack=2801 Win=69760 Len=0 TSval=1529651107 TSecr=2049744091
54	0.810787500	10.11.0.25	216.239.38.120	TCP	66	58250 → 80 [ACK] Seq=79 Ack=5601 Win=75392 Len=0 TSval=1529651107 TSecr=2049744091
55	0.810887287	10.11.0.25	216.239.38.120	TCP	66	58250 → 80 [ACK] Seq=79 Ack=11201 Win=78848 Len=0 TSval=1529651107 TSecr=2049744091
56	0.810942246	216.239.38.120	10.11.0.25	TCP	2866	80 → 58250 [PSH, ACK] Seq=11201 Ack=79 Win=268800 Len=2800 TSval=2049744091 TSecr=1529650615 [TCP PDU reassembled in 6]
57	0.810942836	216.239.38.120	10.11.0.25	TCP	1466	80 → 58250 [ACK] Seq=14001 Ack=79 Win=268800 Len=1400 TSval=2049744125 TSecr=1529650615 [TCP PDU reassembled in 6]
58	0.811005176	10.11.0.25	216.239.38.120	TCP	66	58250 → 80 [ACK] Seq=79 Ack=14001 Win=81664 Len=0 TSval=1529651107 TSecr=2049744091

Rysunek 2. Podgląd komunikacji TCP w aplikacji Wireshark

Time	10.11.0.25	216.239.38.120	Comment
8.304333877	58250	58250 → 80 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK_PERM TSval=1529650600 TSecr=0 WS=128	TCP: 58250 → 80 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK_PERM TSval=1529650600 TSecr=0 WS=128
8.319159675	58250	80 → 58250 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=1412 SACK_PERM TSval=2049743639 TSecr=1529650600 WS=256	TCP: 80 → 58250 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=1412 SACK_PERM TSval=2049743639 TSecr=1529650600 WS=256
8.319278570	58250	58250 → 80 [ACK] Seq=1 Ack=1 Win=64256 Len=0 TSval=1529650615 TSecr=2049743639	TCP: 58250 → 80 [ACK] Seq=1 Ack=1 Win=64256 Len=0 TSval=1529650615 TSecr=2049743639
8.319553379	58250	GET / HTTP/1.1	HTTP: GET / HTTP/1.1
8.337382260	58250	80 → 58250 [ACK] Seq=1 Ack=79 Win=268800 Len=0 TSval=2049743658 TSecr=1529650615	TCP: 80 → 58250 [ACK] Seq=1 Ack=79 Win=268800 Len=0 TSval=2049743658 TSecr=1529650615
8.810539294	58250	80 → 58250 [PSH, ACK] Seq=1 Ack=79 Win=268800 Len=2800 TSval=2049744091 TSecr=1529650615 [TCP PDU reassembled in 6]	TCP: 80 → 58250 [PSH, ACK] Seq=1 Ack=79 Win=268800 Len=2800 TSval=2049744091 TSecr=1529650615 [TCP PDU reassembled in 6]
8.810539931	58250	80 → 58250 [PSH, ACK] Seq=2801 Ack=79 Win=268800 Len=2800 TSval=2049744091 TSecr=1529650615 [TCP PDU reassembled in 6]	TCP: 80 → 58250 [PSH, ACK] Seq=2801 Ack=79 Win=268800 Len=2800 TSval=2049744091 TSecr=1529650615 [TCP PDU reassembled in 6]
8.810540214	58250	80 → 58250 [PSH, ACK] Seq=5601 Ack=79 Win=268800 Len=2800 TSval=2049744091 TSecr=1529650615 [TCP PDU reassembled in 6]	TCP: 80 → 58250 [PSH, ACK] Seq=5601 Ack=79 Win=268800 Len=2800 TSval=2049744091 TSecr=1529650615 [TCP PDU reassembled in 6]
8.810540538	58250	80 → 58250 [PSH, ACK] Seq=8401 Ack=79 Win=268800 Len=2800 TSval=2049744091 TSecr=1529650615 [TCP PDU reassembled in 6]	TCP: 80 → 58250 [PSH, ACK] Seq=8401 Ack=79 Win=268800 Len=2800 TSval=2049744091 TSecr=1529650615 [TCP PDU reassembled in 6]
8.810700524	58250	58250 → 80 [ACK] Seq=79 Ack=2801 Win=69760 Len=0 TSval=1529651107 TSecr=2049744091	TCP: 58250 → 80 [ACK] Seq=79 Ack=2801 Win=69760 Len=0 TSval=1529651107 TSecr=2049744091
8.810787500	58250	58250 → 80 [ACK] Seq=79 Ack=5601 Win=75392 Len=0 TSval=1529651107 TSecr=2049744091	TCP: 58250 → 80 [ACK] Seq=79 Ack=5601 Win=75392 Len=0 TSval=1529651107 TSecr=2049744091
8.810887287	58250	58250 → 80 [ACK] Seq=79 Ack=11201 Win=78848 Len=0 TSval=1529651107 TSecr=2049744091	TCP: 58250 → 80 [ACK] Seq=79 Ack=11201 Win=78848 Len=0 TSval=1529651107 TSecr=2049744091
8.810942246	58250	80 → 58250 [PSH, ACK] Seq=11201 Ack=79 Win=268800 Len=2800 TSval=2049744091 TSecr=1529650615 [TCP PDU reassembled in 6]	TCP: 80 → 58250 [PSH, ACK] Seq=11201 Ack=79 Win=268800 Len=2800 TSval=2049744091 TSecr=1529650615 [TCP PDU reassembled in 6]
8.810942836	58250	80 → 58250 [ACK] Seq=14001 Ack=79 Win=268800 Len=1400 TSval=2049744125 TSecr=1529650615 [TCP PDU reassembled in 6]	TCP: 80 → 58250 [ACK] Seq=14001 Ack=79 Win=268800 Len=1400 TSval=2049744125 TSecr=1529650615 [TCP PDU reassembled in 6]
8.811005176	58250	58250 → 80 [ACK] Seq=79 Ack=14001 Win=81664 Len=0 TSval=1529651107 TSecr=2049744091	TCP: 58250 → 80 [ACK] Seq=79 Ack=14001 Win=81664 Len=0 TSval=1529651107 TSecr=2049744091
8.811048551	58250	58250 → 80 [ACK] Seq=79 Ack=15401 Win=80384 Len=0 TSval=1529651107 TSecr=2049744125	TCP: 58250 → 80 [ACK] Seq=79 Ack=15401 Win=80384 Len=0 TSval=1529651107 TSecr=2049744125
8.826482235	58250	80 → 58250 [PSH, ACK] Seq=15401 Ack=79 Win=268800 Len=2800 TSval=2049744146 TSecr=1529651107 [TCP PDU reassembled in 6]	TCP: 80 → 58250 [PSH, ACK] Seq=15401 Ack=79 Win=268800 Len=2800 TSval=2049744146 TSecr=1529651107 [TCP PDU reassembled in 6]
8.826483077	58250	80 → 58250 [PSH, ACK] Seq=18201 Ack=79 Win=268800 Len=2800 TSval=2049744146 TSecr=1529651107 [TCP PDU reassembled in 6]	TCP: 80 → 58250 [PSH, ACK] Seq=18201 Ack=79 Win=268800 Len=2800 TSval=2049744146 TSecr=1529651107 [TCP PDU reassembled in 6]
8.826483404	58250	HTTP/1.1 200 OK (text/html)	HTTP: HTTP/1.1 200 OK (text/html)

Rysunek 3. Komunikacja TCP w widoku Flow Graph

INDEX: 285358

www.programistamag.pl

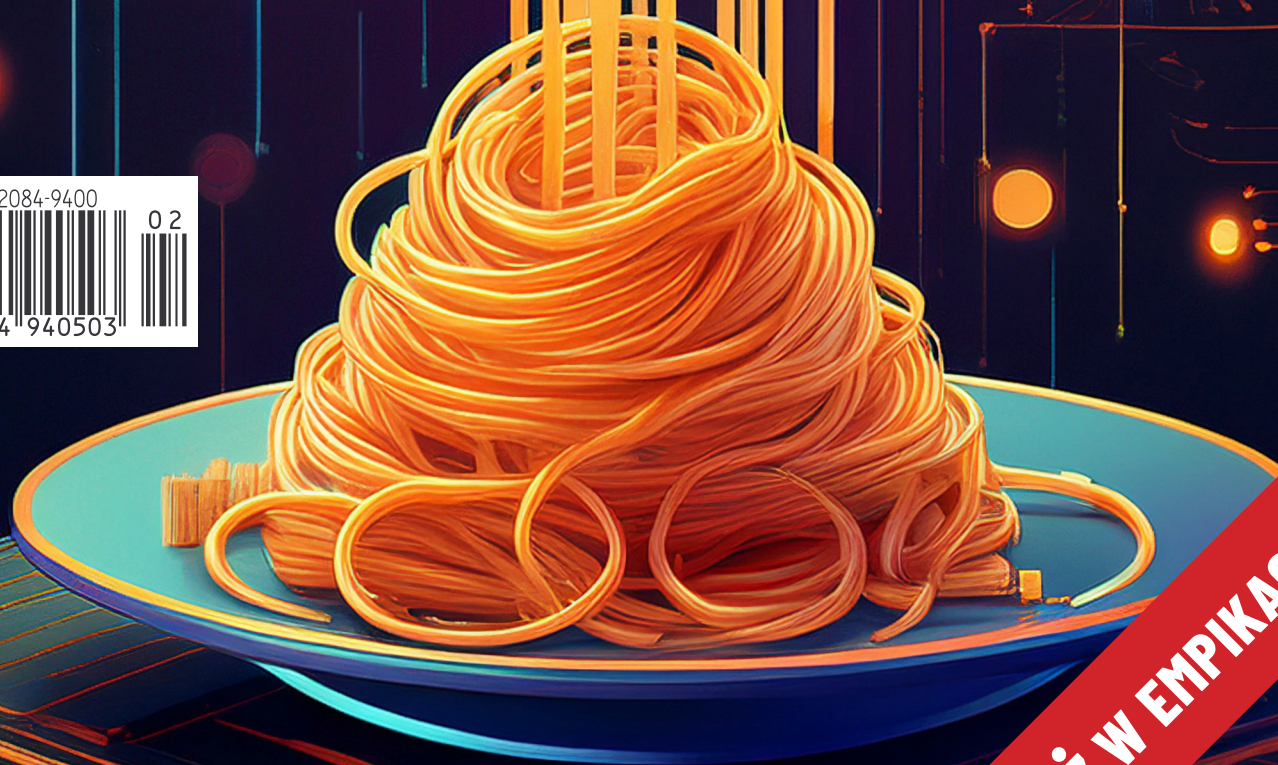
Magazyn programistów i liderów zespołów IT

programista

2/2025 (117)

Cena 32.90 zł (w tym VAT 8%)

CZY „CZYSTY KOD” WYTRZYMAŁ PRÓBĘ CZASU?



JAK DZIAŁA INTERNET - TCP I NAT

WŁASNY SERWER NA VPS

MEGA65 - WSKRZESZONY NASTĘPCA
WIELKIEJ LEGENDY

CIĘŻARNA AUTOMATYZACJA
KONFIGURACJA DLA ASP.NET

NOWY NUMER JUŻ W EMPIKACH