JOURNAL
COMPUTER SCIENCES INSTITUTE

JCSI 19 (2021) 67–74
Received: 13 January 2021
Accepted: 21 January 2021

Comparison of WebSocket and HTTP protocol performance

Porównanie wydajności protokołu WebSocket i HTTP

Wojciech Paweł Łasocha*, Marcin Badurowicz

Department of Computer Science, Lublin University of Technology, Nadbystrzycka 36B, 20-618 Lublin, Poland

Abstract

The purpose of the author of this article is to compare the performance of the WebSocket and HTTP protocols. For this purpose, LAN equipment and a self-made testing application were used. It was used to measure the time of sending and downloading/receiving 100-character texts in a specified number of copies, considering the speed of laptops and web browsers. The conducted research shows that when transmitting more than 100 copies of data using the WebSocket protocol (compared to HTTP), performance can be increased by several hundred percent. In addition, it has been proven that adding excess overhead to HTTP requests can slow it down considerably. In contrast, TLS encryption has little effect on the speed of both protocols. It was concluded that the WebSocket protocol is good for sending hundreds or thousands of small serving of data per second, because for a smaller number of them, a simple HTTP polling is absolutely enough.

Keywords: websocket protocol; http protocol; protocols performance comparison

Streszczenie

Celem autora tego artykułu jest porównanie wydajności protokołu WebSocket i HTTP. W tym celu wykorzystano sprzęt pracujący w sieci LAN oraz samodzielnie wykonaną aplikację testującą. Za jej pomocą zmierzono czas wysyłania oraz pobierania/odbierania 100-znakowych tekstów w określonej liczbie kopii z uwzględnieniem szybkości laptopów i przeglądarek WWW. Z przeprowadzonych badań wynika, że przy transmisji powyżej 100 kopii danych za pomocą protokołu WebSocket (w porównaniu do HTTP) można uzyskać wzrost wydajności o kilkaset procent. Ponadto udowodniono, że dodawanie nadmiarowych narzutów do żądań HTTP może go bardzo spowalniać. Natomiast szyfrowanie TLS ma znikomy wpływ na szybkość obu protokołów. Wywnioskowano, że protokół WebSocket dobrze sprawdzi się w przesyłaniu setek lub tysięcy małych porcji danych na sekundę, gdyż w przypadku mniejszej ich liczby w zupełności wystarczy zwykłe odpytywanie HTTP.

Słowa kluczowe: protokół websocket; protokół http; porównanie wydajności protokołów

*Corresponding author

Email address: wojciech.lasocha@pollub.edu.pl (W. P. Łasocha)

1. Wstęp

Protokół HTTP to jeden z najważniejszych elementów sieci WWW [1]. Umożliwia wymianę danych pomiędzy klientem a serwerem zgodnie z modelem żądanie-odpowiedź [2]. Według artykułu [3] początkowo internautom to w zupełności wystarczało, gdyż byli skoncentrowani na wyszukiwaniu i pobieraniu informacji.

Wraz z rozwojem technologii webowych pojawiała się potrzeba większej interakcji użytkowników na stronach internetowych. Ważnym momentem w rozwoju sieci Web było wprowadzenie techniki AJAX, która pozwalała na asynchroniczną komunikację przeglądarki internetowej z serwerem WWW [4]. Jednak technologia ta nadal bazowała na modelu żądanie-odpowiedź. Programiści aplikacji internetowych próbowali ją wykorzystać do emulacji komunikacji w czasie rzeczywistym, poprzez zastosowanie długiego odpytywania HTTP (ang. *Long Polling HTTP*). Mimo, że metoda ta dalej jest używana w całym Internecie, nadal nie rozwiązuje tego problemu, że dane mogą zostać przesłane jedynie podczas zainicjowania żądania przez przeglądarkę – w tym wypadku nie ma możliwości bezpośredniego wysłania informacji przez serwer do klienta bez jego wcześniejszego żądania o nie. Ponadto, technika ta może powodować poważne problemy, ze względu na generowane duże obciążenia zarówno przeglądarki klienta, jak i serwera [5].

Jak wyjaśniają artykuły [6-7], przełomowym momentem było wprowadzenie w 2008 r. mechanizmu WebSocket. Odzwierciedla on sieć WWW czasu rzeczywistego (ang. *Real-Time Web*), która według artykułu [8] umożliwia użytkownikom otrzymywanie informacji natychmiast po ich opublikowaniu przez ich autorów, zamiast wymagać od nich okresowego sprawdzania źródła pod kątem aktualizacji. Znakomicie zmniejsza to obciążenie sieci, serwera i samej przeglądarki. Unika się dodatkowych komunikatów przesyłanych i przetwarzanych po obu stronach tylko dla utrzymania stałej komunikacji oraz umożliwia serwerowi przesyłanie do przeglądarki w czasie rzeczywistym nowych danych, kiedy tylko pojawią się na serwerze. Rozwiązania te mają korzystny wpływ na wydajność.

Według autora artykułu [5] wykonanie pojedynczego żądania na połączenie z wykorzystaniem biblioteki WebSocket Socket.IO jest o połowę wolniejsze niż w przypadku protokołu HTTP, ponieważ połączenie musi zostać najpierw ustanowione. Natomiast wykonanie 50 żądań w ramach tego samego połączenia poprzez WebSocket jest o połowę szybsze niż używając do tego

celu HTTP. W tym samym artykule można również przeczytać, że HTTP osiąga szczytową przepustowość przy około ~950 żądaniach na sekundę, podczas gdy Socket.IO obsługuje około ~3900 żądań na sekundę.

Badania przeprowadzone przez autorów artykułu [9] dowodzą, że protokół WebSocket jest nieco szybszy niż HTTP. Wysyłanie na serwer 40000 bajtów danych w sieci WAN za pomocą techniki AJAX trwało około 39 ms, natomiast protokół WebSocket zrobił to samo w około 29 ms. Wysyłanie tych samych danych w sieci LAN wyszło również korzystniej dla tej technologii i było o 4 ms szybsze.

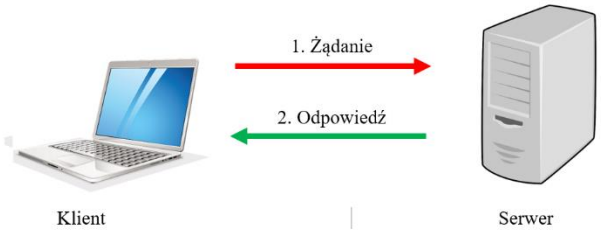
Natomiast autor badań opisanych w artykule [10] udowodnił, że zastosowanie bardzo szybkich serwerów (w szczególności ich procesorów) pozwala na obsługę 5 milionów jednocześnie podłączonych klientów z wykorzystaniem protokołu WebSocket.

Wyniki badań w powyższych artykułach [5, 9-10] motywują do przeprowadzenia własnych badań nad wydajnością obu protokołów. Autor niniejszego artykułu skoncentrował się na zmierzeniu szybkości częstego (wielokrotnego) przesyłania małych porcji danych, transmitowanych w setkach, a nawet tysiącach komunikatów na sekundę. Dlatego przedstawione wyniki opisują stan rzeczy, jak dany protokół sprawdzi się w takich zastosowaniach jak: gry online, pokoje czatowe, czy pobieranie w czasie rzeczywistym notowań giełdowych. Rozważanie przypadku przesyłania plików o dużych rozmiarach jest zbędne, gdyż protokół WebSocket po prostu do tego się nie nadaje ze względu na fragmentację danych na części po kilkadziesiąt kilobajtów każda. Dlatego przesyłanie pliku o rozmiarze 1 MB może trwać tyle samo czasu dla HTTP, jak i dla WebSocket.

2. Przedmiot badań

2.1. Charakterystyka protokołu HTTP

Protokół HTTP tworzy kanał wymiany danych pomiędzy dwoma końcami komunikacji, którymi najczęściej są klient (na przykład przeglądarka internetowa) i serwer (na przykład aplikacja działająca na komputerze hostującym witrynę internetową). Protokół ten bazuje na modelu żądanie-odpowiedź, tzn. klient wysyła do serwera komunikat żądania, po czym serwer zwraca komunikat odpowiedzi zawierający plik HTML lub inny zasób. Odpowiedź zawiera informację o statusie ukończenia wykonanego żądania i może również zawierać żądaną treść w ciele wiadomości (na przykład strukturę pliku HTML). Graficzny model wymiany danych pomiędzy klientem a serwerem z wykorzystaniem protokołu HTTP został przedstawiony na rysunku numer 1.



Rysunek 1: Model wymiany danych pomiędzy klientem a serwerem z wykorzystaniem protokołu http.

Do zadań klienta HTTP zalicza się: wysyłanie żądania – inicjowanie połączenia HTTP (poprzez URL), pobieranie zasobu z serwera, prezentacja danych (budowa strony internetowej), interakcja z użytkownikiem, buforowanie danych z serwera oraz kontrola spójności z serwerem, szyfrowanie-deszyfrowanie (HTTPS).

Natomiast do zadań serwera HTTP zalicza się: obsługa żądań HTTP – uruchamianie skryptów i wysyłanie odpowiedzi, rejestracja i kolejkowanie żądań, uwierzytelnianie i kontrola dostępu, szyfrowanie-deszyfrowanie (HTTPS), wybór wersji językowej wysyłanych dokumentów.

HTTP to protokół aplikacji, znajdujący się w warstwie 7. modelu OSI/ISO. Jest oparty na TCP (protokół warstwy transportowej) i domyślnie wykorzystuje port 80 (dla połączeń nieszyfrowanych) lub 443 (dla połączeń szyfrowanych). Chociaż, że może przesyłać nie tylko dane tekstowe, ale również binarne, na ogół uważany jest za tekstowy, ponieważ używa znakowych poleceń i komunikatów. Ważną jego cechą jest to, że zalicza się do protokołów bezstanowych, ponieważ nie zachowuje żadnych informacji o poprzednich transakcjach z klientem (po zakończeniu transakcji wszystko zostaje „zapominane"). Zalety takiego rozwiązania to lepsze wykorzystanie zasobów (obiekt po stronie serwera obsługujący żądanie po wysłaniu danych może rozpocząć obsługę kolejnego żądania) oraz prostota migracji ruchu na inne serwery. Znacznie to zmniejsza obciążenie serwera, jednak jest utrudnieniem w sytuacji, gdy na przykład trzeba zapamiętać konkretny stan dla użytkownika, który wcześniej już łączył się z serwerem. Programiści aplikacji internetowych zwykle rozwiązują ten problem poprzez wykorzystanie mechanizmu ciasteczek lub sesji po stronie serwera, czy wprowadzając ukryte parametry (w formularzu lub adresie URL).

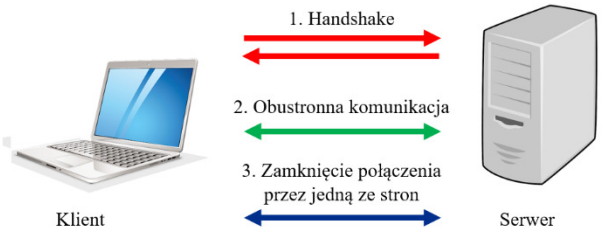
2.2. Charakterystyka protokołu WebSocket

Jak można przeczytać w artykule [6] WebSocket to komputerowy protokół komunikacyjny, zapewniający kanały komunikacji za pośrednictwem jednego połączenia TCP pomiędzy klientem (na przykład przeglądarką internetową) a serwerem. Istotne jest to, że zapewnia wymianę danych w czasie rzeczywistym, tzn. serwer może wysłać do klienta wiadomość bez wcześniejszego żądania klienta o nią. W ten sposób może odbywać się dwukierunkowa trwająca rozmowa między klientem a serwerem, utrzymując połączenie otwarte. W przeciwieństwie do protokołu HTTP, protokół WebSocket umożliwia przesyłanie danych w trybie pełnego dupleksu, czyli może jednocześnie wysyłać i odbierać dane. Protokół ten ma znacznie mniejsze narzuty i nie wysyła zbędnych komunikatów jak w przypadku odpytywania HTTP (ang. *HTTP Polling*), przez co zmniejszają się opóźnienia w komunikacji, obciążenie sieci, serwera oraz samej przeglądarki internetowej, co ma korzystny wpływ na wydajność.

Protokół WebSocket jest ogromnym krokiem naprzód, pozwalającym na tworzenie aplikacji czasu rzeczywistego opartego na zdarzeniach. Można rzec, że WebSocket daje możliwości, które w przypadku proto-

kołu HTTP, były praktycznie nieosiągalne. Dzięki niemu klient nie musi prosić serwera o aktualne dane, lecz otrzyma je automatycznie w chwili ich pojawienia się na serwerze. Możliwości jego zastosowania są ogromne, na przykład obserwowanie cen akcji, informacji prasowych, sprzedaży biletów, natężenia ruchu, odczytów z urządzeń medycznych, na żywo. Należy jednak wspomnieć, że protokół HTTP przeznaczony jest do przesyłania danych tekstowych i binarnych, natomiast protokół WebSocket służy do szybkiej wymiany danych w formacie JSON (ang. *JavaScript Object Notation*). Przeglądarki internetowe potrafią wyświetlić stronę WWW na podstawie kodu HTML. Jednak obecnie nie ma takiego standardu, który mógłby zrobić to samo analizując dane JSON. Z tego względu można stwierdzić, że protokół WebSocket stanowi niejako dodatek do obecnej sieci Web, a nie jest następcą tradycyjnego protokołu HTTP.

Zanim protokół WebSocket rozpocznie wymianę danych pomiędzy klientem a serwerem, klient wysyła do serwera wiadomość w postaci zwykłego żądania HTTP z prośbą o zmianę protokołu i nawiązanie stałego połączenia – to tzw. uścisk dłoni (ang. *Handshake*). Jeśli serwer zaakceptuje ją, wysyła do klienta wiadomość zwrotną z pozytywną odpowiedzią. W tym momencie obie strony komunikacji mogą wysyłać do siebie dane. Oczywiście, w każdym momencie połączenie może być jawnie zamknięte. Odłączenie klienta lub serwera zawsze zostanie wykryte i o fakcie tym może zostać poinformowany użytkownik. Graficzny model wymiany danych pomiędzy klientem a serwerem z wykorzystaniem protokołu WebSocket został przedstawiony na rysunku numer 2.



Rysunek 2: Model wymiany danych pomiędzy klientem a serwerem z wykorzystaniem protokołu WebSocket.

WebSocket, podobnie jak protokół HTTP, to protokół aplikacji znajdujący się w warstwie 7. modelu OSI/ISO, oparty na TCP (protokół warstwy transportowej). WebSocket, choć różni się od HTTP, jest zaprojektowany do pracy przez porty HTTP 80 (dla połączeń nieszyfrowanych) i 443 (dla połączeń szyfrowanych), dlatego jest z nim zgodny. Jest zdolny do pracy przez istniejącą infrastrukturę przystosowaną do transmisji HTTP, czyli przez już skonfigurowane do tego celu serwery pośredniczące (ang. *Proxy*) oraz przechodził przez ustawienia zapory sieciowej, jeżeli tylko dopuszczany jest ruch HTTP.

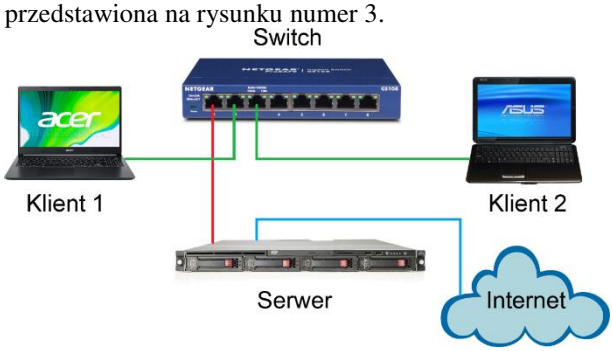
3. Metodyka badań

Do wykonania badań nad wydajnością protokołów HTTP i WebSocket autor niniejszego artykułu zdecydował się zbudować własne środowisko badawcze. System ten składa się z następujących elementów:

- Infrastruktura – fizyczna część systemu składająca się z serwera, przełącznika niezarządzalnego (ang. *Switch*) i dwóch komputerów klienckich, połączonych kablem w sieć lokalną LAN (ang. *Local Area Network*) za pomocą skrętki czteroparowej.
- Bazowe oprogramowanie klienckie – dwa odrębne środowiska zainstalowane na komputerach klienckich, w skład których zawierają się systemy operacyjne Microsoft Windows 10 i Linux Fedora 33 Workstation oraz przeglądarki internetowe Google Chrome, Mozilla Firefox, Microsoft Edge.
- Bazowe oprogramowanie serwerowe – środowisko zainstalowane na serwerze, oparte na systemie operacyjnym Debian 10 (Buster), platformie Node.js oraz bibliotek Express.js (ułatwiającej zbudowanie serwera WWW wykorzystującego protokół HTTP) i Socket.IO (ułatwiającej tworzenie aplikacji komunikujących się za pomocą protokołu WebSocket).
- Aplikacja testująca – samodzielnie wykonana internetowa aplikacja webowa (uruchamiana w przeglądarce internetowej) oparta na platformie Node.js dzieląca się na część serwerową i kliencką, mierząca czas transmisji danych o określonych parametrach za pomocą protokołów HTTP i WebSocket.

3.1. Wykorzystana infrastruktura

Struktura zbudowanej przewodowej sieci lokalnej wraz z połączeniem ze sobą wszystkich urządzeń została przedstawiona na rysunku numer 3.



Rysunek 3: Struktura zbudowanej przewodowej sieci lokalnej wykorzystanej w badaniach.

Centralnym punktem wiążącym sieć jest przełącznik. Do jego pierwszego portu podłączono serwer, natomiast do drugiego i trzeciego portu podłączono komputery klienckie. Drugi interfejs sieciowy serwera łączy się z Internetem. Poszczególne wykorzystane urządzenia charakteryzują się następującymi parametrami:

- Switch Netgear GS108GE
  - 8 portów Gigabit Ethernet
  - Niezarządzalny
  - Brak PoE (ang. *Power over Ethernet*)
- Serwer HP ProLiant DL320 G5
  - System operacyjny Linux Debian 10 (Buster)
  - Procesor Quad-Core Intel Xeon 2,4 GHz
  - Pamięć operacyjna DDR2-800 6 GB
  - 4 dyski twarde SAS 15000 po 72 GB każdy
  - 2 interfejsy sieciowe Gigabit Ethernet

- Laptop 1 Acer Aspire 5
  - System operacyjny Microsoft Windows 10
  - Procesor Intel Core i5-7200U 2,5 GHz
  - Pamięć operacyjna DDR4-2133 4 GB
  - Dysk twardy SATA III 5400 1000 GB
  - Interfejs sieciowy Gigabit Ethernet
- Laptop 2 Asus K50IN
  - System operacyjny Linux Fedora 33 Workstation
  - Procesor Intel Pentium Dual Core T4200 2 GHz
  - Pamięć operacyjna DDR2-800 4 GB
  - Dysk twardy SATA I 5400 60 GB
  - Interfejs sieciowy Gigabit Ethernet
- Okablowanie
  - Skrętka czteroparowa
  - Kategoria 6

Serwer posiada cztery dyski twarde połączone w macierz dyskową RAID 0. Z tego względu można z góry wykluczyć, że wąskim gardłem systemu był zapis i odczyt danych wysyłanych do serwera bądź z niego pobieranych. Transmisję danych mógł jedynie spowalniać komputer kliencki Asus K50IN (ponieważ jest to stary sprzęt), co oczywiście było uwzględnione w testach. Aczkolwiek, zastosowanie Gigabitowych interfejsów sieciowych w każdym z urządzeń, powinno dać możliwie jak najbardziej realistyczne wyniki przeprowadzanych badań.

3.2. Instalacja i konfiguracja oprogramowania

Do wykonania badań, autor niniejszego artykułu wykorzystał odpowiednie oprogramowanie – zarówno dla komputerów klienckich, jak i serwera.

Na nowszym i szybszym laptopie Acer Aspire 5 został wykorzystany system operacyjny Microsoft Windows 10, natomiast na starszym i wolniejszym Asus K50IN – Linux Fedora 33 Workstation. Do tego pierwszego domyśle jest dołączana przeglądarka internetowa Microsoft Edge, a do drugiego Mozilla Firefox. Jednak w testach została użyta również przeglądarka WWW Google Chrome, która nie była dołączona do żadnego OS-u. Dlatego brakujące programy wykorzystane w testach musiały być ręcznie doinstalowane.

Najważniejszym oprogramowaniem wykorzystanym na serwerze HP ProLiant DL320 G5 był system operacyjny Linux Debian 10 (Buster), który z reguły przeznaczony jest do zastosowań serwerowych. Po jego zainstalowaniu, interfejsy sieciowe serwera zostały tak skonfigurowane, aby jedna karta sieciowa miała dostęp do Internetu, druga zaś była rozpoznawalna w sieci lokalnej, do której były podłączone komputery klienckie. Aby zaoszczędzić sobie pracy w konfiguracji sieci na klientach, na serwerze został skonfigurowany serwer DHCP. Następnie została zainstalowana platforma Node.js wraz z menadżerem pakietów npm oraz dodatkowo narzędzie OpenSSL. Te ostatnie służy do wygenerowania certyfikatu CSR i klucza prywatnego, potrzebnych do zestawiania połączeń szyfrowanych SSL/TLS (HTTPS oraz WebSocket Secure).

Po wykonaniu powyższych czynności, wszystkie urządzenia (a w pierwszej kolejności serwer) zostały ponownie uruchomione.

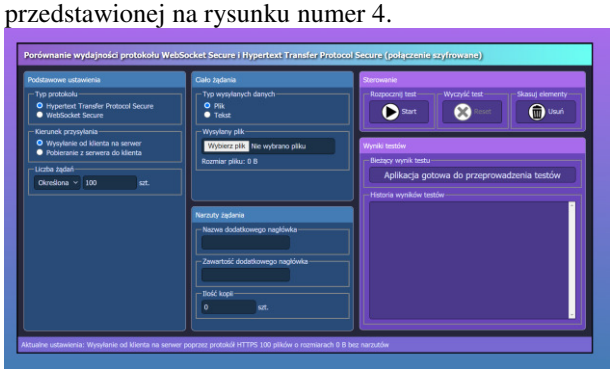
3.3. Aplikacja testująca

W celu zainstalowania aplikacji testującej na serwerze należało skopiować rekurencyjnie jej główny katalog na serwer, następnie przejść do niego i wydać polecenie `npm install`. Potem konieczne było zmodyfikowanie zawartości pliku `settings.txt` znajdującego się w katalogu `public` na adres IP interfejsu sieciowego serwera pod jakim ma być rozpoznawalny w sieci LAN, do której były podłączone komputery klienckie.

W celu uruchomienia aplikacji testującej na serwerze należało przejść do jej głównego katalogu, a następnie w zależności czy do połączeń ma być wykorzystywane szyfrowanie TLS, czy też nie, wykonano jedno z dwóch poniższych poleceń:

- Dla połączeń nieszyfrowanych:
  `node index.js --encryption=no`
- Natomiast dla połączeń szyfrowanych:
  `node index.js --encryption=yes`

Od tego momentu na wszystkich podłączonych klientach była dostępna aplikacja (od strony klienckiej) w przeglądarkach internetowych pod adresem URL równym adresowi IP serwera, poprzedzonym prefiksem używanego protokołu (`http://` lub `https://`), w zależności od tego, czy zostało zastosowane szyfrowanie w przesyłaniu danych. Po otwarciu strony internetowej ukarze się główny interfejs aplikacji testującej, przedstawionej na rysunku numer 4.



Rysunek 4: Główny interfejs użytkownika aplikacji testującej otwartej w przeglądarce internetowej.

Interfejs użytkownika aplikacji testującej przedstawionej na powyższym rysunku składa się z dwóch najważniejszych części. Po jego lewej stronie (ramki w kolorze niebieskim) użytkownik może dowolnie dostosować ustawienia testu, który ma zostać przeprowadzony. Zmiany tychże ustawień są na bieżąco wyświetlane w pasku stanu (znajdującego się na dole aplikacji testującej), który zwięźle i kompleksowo opisuje aktualnie wybrane ustawienia. Po prawej stronie interfejsu aplikacji (ramki w kolorze fioletowym) użytkownik może sterować testem oraz analizować jego wyniki po ukończeniu badań. Wybór ustawień testu sprowadza się do typu protokołu (HTTP/HTTPS lub WebSocket/WebSocket Secure), kierunku przesyłania (wysyłanie na serwer, pobieranie lub odbieranie z serwera, czy transfer pomiędzy klientami), rodzaju przesyłanych danych (plik lub tekst) oraz liczby ich kopii, a także opcjonalnych narzutów (w przypadku protokołu HTTP/HTTPS).

Po skonfigurowaniu aplikacji testującej, można rozpocząć badanie klikając przycisk `Start`. Na ekranie serwera na bieżąco pojawiają się postępy w jego wykonaniu, jak dla przykładu pokazano na rysunku numer 5 – w tym przypadku wysyłanie od klienta na serwer poprzez szyfrowany protokół WSS (ang. *WebSocket Secure*) 10 tekstów o długościach 16 znaków każdy. Końcowy wynik wykonania operacji jest zawsze wyświetlany zarówno w aplikacji klienckiej, jak i serwerowej.

```
10:45:27,852 - Zlecono: Wysyłanie od klienta na serwer poprzez protokół WSS
10 tekstów o długościach 16 znaków
10:45:27,855 - Pomyślnie odebrano i zapisano tekst na dysku twardym serwera
(całkowity postęp: 10,00%)
10:45:27,857 - Pomyślnie odebrano i zapisano tekst na dysku twardym serwera
(całkowity postęp: 20,00%)
10:45:27,858 - Pomyślnie odebrano i zapisano tekst na dysku twardym serwera
(całkowity postęp: 30,00%)
10:45:27,859 - Pomyślnie odebrano i zapisano tekst na dysku twardym serwera
(całkowity postęp: 40,00%)
10:45:27,861 - Pomyślnie odebrano i zapisano tekst na dysku twardym serwera
(całkowity postęp: 50,00%)
10:45:27,862 - Pomyślnie odebrano i zapisano tekst na dysku twardym serwera
(całkowity postęp: 60,00%)
10:45:27,864 - Pomyślnie odebrano i zapisano tekst na dysku twardym serwera
(całkowity postęp: 70,00%)
10:45:27,866 - Pomyślnie odebrano i zapisano tekst na dysku twardym serwera
(całkowity postęp: 80,00%)
10:45:27,866 - Pomyślnie odebrano i zapisano tekst na dysku twardym serwera
(całkowity postęp: 90,00%)
10:45:27,867 - Pomyślnie odebrano i zapisano tekst na dysku twardym serwera
(całkowity postęp: 100,00%)
10:45:27,867 - Pomyślnie ukończono wysyłanie tekstów od klienta na serwer (w
ysłano 10 tekstów w czasie 0,015 sekund)
```

Rysunek 5: Ekran konsoli Linuksa na serwerze po pomyślnie wykonanym teście.

Klikając przyciski `Reset` i `Start` można kontynuować badania i analizować wyniki, a w razie konieczności zmieniać parametry testów. Pamiętać również trzeba, aby przed każdym wysyłaniem danych na serwer usunąć z niego elementy, które były wysyłane do niego poprzednim razem, aby uniknąć kolizji wśród plików – można to zrobić klikając przycisk `Usuń`. Niestety, aby włączyć lub wyłączyć szyfrowanie w przesyłanych danych, należy od nowa uruchomić aplikację – zarówno od strony serwerowej, jak i klienckiej. Aplikację serwerową można zakończyć naciskając kombinację klawiszy `Ctrl + C`.

4. Wyniki przeprowadzonych badań

4.1. Wyjaśnienia wstępne

Badania przeprowadzone przez autora niniejszego artykułu polegają na testach przesyłania danych poprzez protokoły HTTP i WebSocket, a dokładniej zmierzeniu czasów wysyłania i pobierania (lub odbierania w przypadku WebSocket) krótkich tekstów. Każdy wspomniany tekst ma rozmiar około 100 bajtów, gdyż składa się z ciągu 100 znaków, a jego dokładna treść to:

```
1234567890123456789012345678901234567
8901234567890123456789012345678901234
5678901234567890123456789
```

Mimo, że powyższy tekst ma stały i z góry określony rozmiar, wykonane badania koncentrują się na porównaniu czasów przesyłania danych w różnych ilościach kopii, tzn. liczby kolejno następujących po sobie żądań-odpowiedzi (dla protokołu HTTP) lub wiadomości (dla protokołu WebSocket). W pomiarach uwzględniono przypadki dla 1, 3, 10, 30, 100, 300, 1000 i 3000 kopii, a wyniki przedstawiono w formie wykresów z dokładnością co do milisekundy (ms).

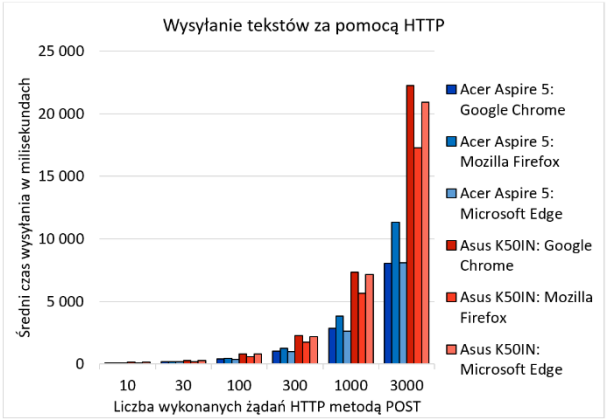
W badaniach każda próba została wykonana 10-krotnie, a na koniec obliczono z nich wartość średnią. W testach uwzględniono szybkość komputerów klienckich oraz wydajność zainstalowanych na nich systemów operacyjnych i przeglądarek internetowych, gdyż każdy test został przeprowadzony odrębnie dla danej platformy klienckiej.

Należy dodać, że badania zostały przeprowadzone w pełni domyślnych ustawieniach protokołów, stąd można uzyskać o wiele gorsze wyniki dla protokołu HTTP, jeśli zwiększy się jego narzuty (co zostanie udowodnione w dalszej części tego artykułu), lecz rzadko się to robi. Jednocześnie, protokół WebSocket potrafi czasami zaskakiwać, gdyż w badaniach do odebrania z serwera 10 wiadomości potrzebował około 90 ms, a zdarzało się, że robił to w 15 ms.

4.2. Wysyłanie i pobieranie poprzez HTTP

Pierwszym etapem niniejszych prac było zbadanie szybkości przesyłania 100-znakowych tekstów za pomocą protokołu HTTP.

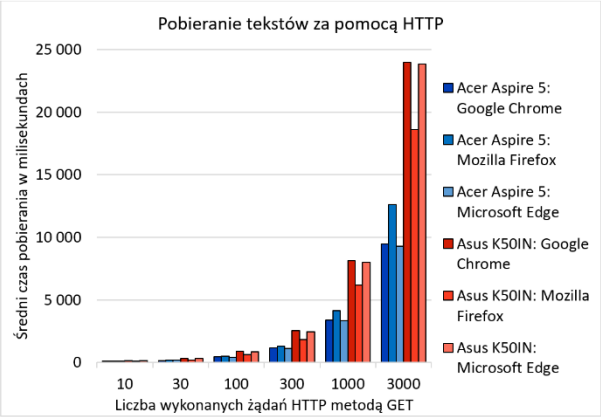
W pierwszym kroku zmierzono czas wysyłania danych z komputerów klienckich na serwer za pomocą metody POST. Następnie wykonano wykres słupkowy ze średnich wartości, który przedstawiono na rysunku numer 6.



Rysunek 6: Średnie czasy wysyłania tekstów z komputerów klienckich na serwer za pomocą protokołu http.

Na powyższym wykresie wyraźnie widać, że wykonywanie większej liczby żądań HTTP (powyżej 30) metodą POST i przetwarzanie ich odpowiedzi, szybciej przebiegło w przypadku komputera klienckiego Acer Aspire 5 niż z Asus K50IN, gdyż jest to o wiele lat nowszy sprzęt (i co oczywiste szybszy). Co ciekawe, można również zauważyć dużą różnicę w wydajności przeglądarki internetowej Mozilla Firefox w zależności od platformy na jakiej pracuje (nie wiadomo jednak, czy zależy to od szybkości komputera, czy od systemu operacyjnego: Windows 10 lub Fedora 33). Można jednak przepuszczać, że najlepiej sprawdzają się przeglądarki WWW w systemach operacyjnych domyślnie do nich dołączane (Edge dla Windows oraz Firefox dla Fedory).

W drugim kroku zmierzono czas pobierania danych z serwera na komputery klienckie za pomocą metody GET. Następnie wykonano wykres słupkowy ze średnich wartości, który przedstawiono na rysunku numer 7.



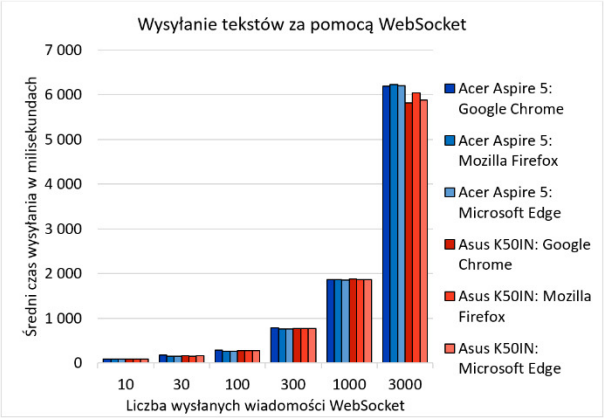
Rysunek 7: Średnie czasy pobierania tekstów z serwera przez komputery klienckie za pomocą protokołu http.

Na powyższym wykresie można zauważyć większą szybkość wykonywania żądań HTTP metodą GET w przypadku szybszego komputera klienckiego. Wystąpił również wzrost wydajności przeglądarki internetowej Mozilla Firefox (w porównaniu do pozostałych przeglądarek WWW) w zależności od laptopa i systemu operacyjnego na którym została użyta. Można stwierdzić, że program Microsoft Edge bez względu na szybkość sprzętu tak samo dobrze radził sobie z pobieraniem danych wykonując żądania HTTP GET jak Google Chrome.

4.3. Wysyłanie i odbieranie poprzez WebSocket

Drugim etapem niniejszych prac było zbadanie szybkości przesyłania 100-znakowych tekstów za pomocą protokołu WebSocket.

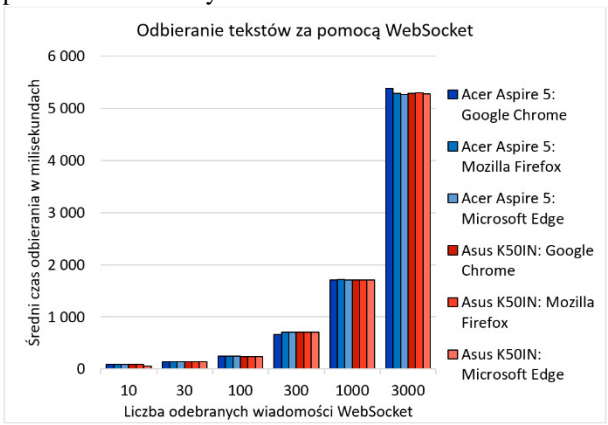
W pierwszym kroku zmierzono czas wysyłania danych z komputerów klienckich na serwer. Następnie wykonano wykres słupkowy ze średnich wartości, który przedstawiono na rysunku numer 8.



Rysunek 8: Średnie czasy wysyłania tekstów z komputerów klienckich na serwer za pomocą protokołu WebSocket.

Najbardziej co wrzuca się w oczy to fakt, że wydajność protokołu WebSocket nie zależy od wykorzystywanego oprogramowania (systemów operacyjnych i przeglądarek internetowych), a nawet od szybkości komputerów klienckich. Z tego względu programiści aplikacji wykorzystujących protokół WebSocket nie muszą martwić się o użytkowników klienckich z powolnym sprzętem, czy rzadko używanym oprogramowaniem.

W drugim kroku zmierzono czas odbierania danych z serwera przez komputery klienckie. Następnie wykonano wykres słupkowy ze średnich wartości, który przedstawiono na rysunku numer 9.

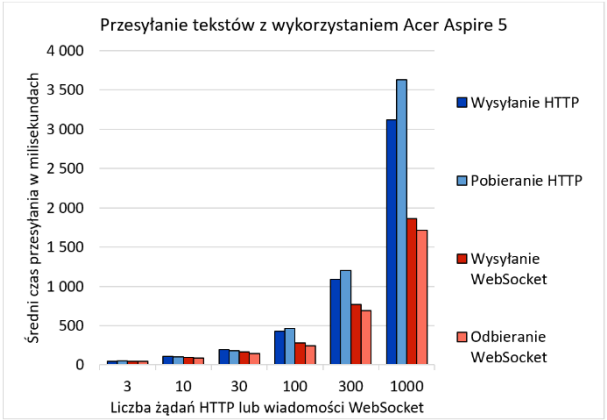


Rysunek 9: Średnie czasy odbierania tekstów z serwera przez komputery klienckie za pomocą protokołu WebSocket.

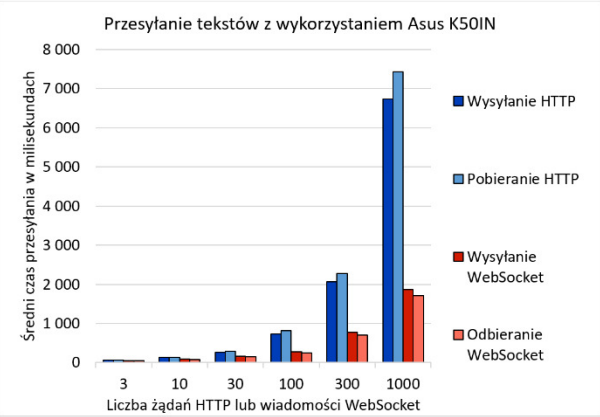
Na powyższym wykresie można dostrzec brak wpływu oprogramowania bazowego (systemów operacyjnych i przeglądarek internetowych) oraz sprzętu klienckiego na wydajność protokołu WebSocket.

4.4. Porównanie wydajności HTTP i WebSocket

W tej części artykułu podjęto próbę porównania wydajności protokołów HTTP i WebSocket poprzez analizę ich szybkości wysyłania i pobierania (odbierania w przypadku WebSocket) tekstów. Porównania dokonano na podstawie wyników badań uzyskanych w dwóch poprzednich podrozdziałach, kategoryzując je według sprzętu (szybszego i wolniejszego laptopa), na których zostały użyte, nie uwzględniając jednak przeglądarek internetowych (tzn. uśredniając uzyskane przez nie wyniki). W tym celu wykonano odpowiednie wykresy słupkowe przedstawione na rysunkach od numeru 10 do numeru 11.



Rysunek 10: Średnie czasy przesyłania tekstów pomiędzy komputerem klienckim Acer Aspire 5 a serwerem za pomocą protokołów HTTP i WebSocket.



Rysunek 11: Średnie czasy przesyłania tekstów pomiędzy komputerem klienckim Asus K50IN a serwerem za pomocą protokołów HTTP i WebSocket.

Najbardziej co wrzuca się w oczy, patrząc na powyższe wykresy to fakt, że protokół WebSocket jest znacznie szybszy od HTTP w przypadku wysyłania lub pobierania (odbierania) powyżej 100 kopii danych (żądań-odpowiedzi dla HTTP lub wiadomości dla WebSocket). Różnica ta w szczególności jest większa dla wolniejszego laptopa, gdyż jak wcześniej zauważono, wydajność protokołu WebSocket jest niezależna od platformy klienckiej. Tym samym można stwierdzić, że nie ma sensu implementować technologii WebSocket dla przesłania kilku lub kilkunastu wiadomości – dlatego w tym przypadku wystarczy zastosować zwykłe odpytywanie HTTP – tym bardziej, że powstają coraz szybsze komputery, które coraz bardziej sprawnie wykonują żądania HTTP i obsługują jego odpowiedzi. Tak więc protokół WebSocket świetnie sprawdzi się w aplikacjach sieciowych, gdzie wymagane jest przesyłanie dużej ilości (setek, a nawet tysięcy) małych porcji danych na sekundę.

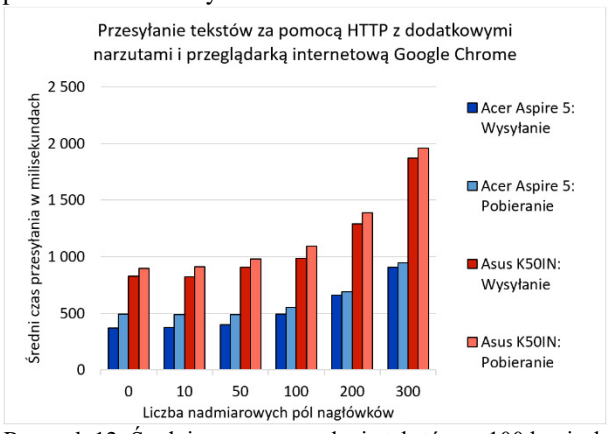
Najogólniej i powszechnie uważa się, że nie ma różnicy w wydajności pomiędzy pobieraniem danych z serwera za pomocą metody GET protokołu HTTP, a ich wysyłaniem za pomocą metody POST. Jednak jak wynika z badań, dla większej ilości zapytań HTTP, tj. 100 i więcej można zauważyć większą szybkość w przypadku tej drugiej metody. Wynika to z faktu, że komputery klienckie (a dokładniej przeglądarki internetowe) nie są w stanie na raz obsłużyć tak dużej ilości odpowiedzi HTTP GET z serwera.

W przypadku protokołu WebSocket odbieranie danych z serwera zawsze przebiegało trochę szybciej niż ich wysyłanie na serwer lub przekazywanie do innych klientów. Prawdopodobnie na werdykt ten wpłyną fakt, iż technologia ta nie wymaga od nich wysokiej wydajności, lecz jest zależna od mocy obliczeniowej serwera, na którym została użyta.

Ostatecznie, stosując protokół WebSocket w przesyłaniu dużej liczby kopii niewielkich danych w zadanym czasie można uzyskać wzrost wydajności nawet o kilkaset procent. Można nawet przewidywać, że za jego pomocą przesłanie kilku lub jednej kopii takich danych (w porównaniu do zwykłego protokołu HTTP) może być nieco szybsze.

4.5. Wpływ narzutów na szybkość przesyłania

Kolejnym etapem niniejszych prac było zbadanie szybkości wysyłania 100 kopii danych z komputerów klienckich na serwer za pomocą metody POST oraz ich pobieraniem z serwera za pomocą metody GET wraz z określonymi narzutami żądania HTTP. W badaniach każde żądanie HTTP (spośród 100 wykonanych) wzbogacono o daną liczbę dodatkowych pól nagłówków w formacie:
`nazwa-1: zawartosc`, `nazwa-2: zawartosc`, itd. Testy obejmowały przypadki dodania 10, 50, 100, 200 i 300 nadmiarowych narzutów. Należy nadmienić, że technika ta jest możliwa jedynie w przypadku protokołu HTTP – protokół WebSocket uniemożliwia wykonywanie takich czynności. Na podstawie wyników wykonano wykres słupkowy ze średnich wartości, który przedstawiono na rysunku numer 12.



Rysunek 12: Średnie czasy przesyłania tekstów w 100 kopiach pomiędzy komputerami klienckimi a serwerem za pomocą HTTP z dodatkowymi narzutami.

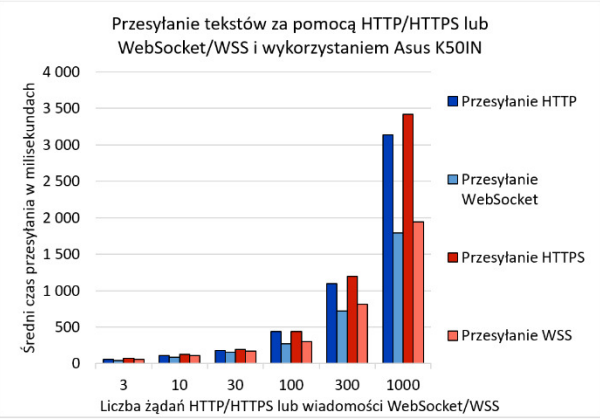
Analizując uzyskane wyniki, można stwierdzić, że dodanie kilku lub kilkunastu dodatkowych pól nagłówków nie ma znaczącego wpływu na wydajność protokołu HTTP. Jeśli jednak ich liczba wyniesie 100, co zazwyczaj rzadko się zdarza, szybkość przesyłania tekstów – według przeprowadzonych badań – może spaść o około 20%. W tym przypadku rozmiar nadmiarowych narzutów w każdym żądaniu HTTP wyniesie około 1900 znaków (bajtów), stąd ten wynik. Należy podkreślić, że w testach uwzględniono dodawanie dodatkowych narzutów dla każdego żądania HTTP, bo jeśli zostaną one dodane również do odpowiedzi HTTP, wynik będzie jeszcze gorszy.

4.6. Wpływ szyfrowania na szybkość przesyłania

Ostatnim etapem niniejszych prac było zbadanie szybkości przesyłania tekstów poprzez protokół HTTPS (czyli poprzez HTTP z szyfrowaniem TLS) oraz protokół WebSocket Secure (czyli poprzez WebSocket z szyfrowaniem TLS – w skrócie WSS) oraz porównano je z zwykłymi odpowiednikami.

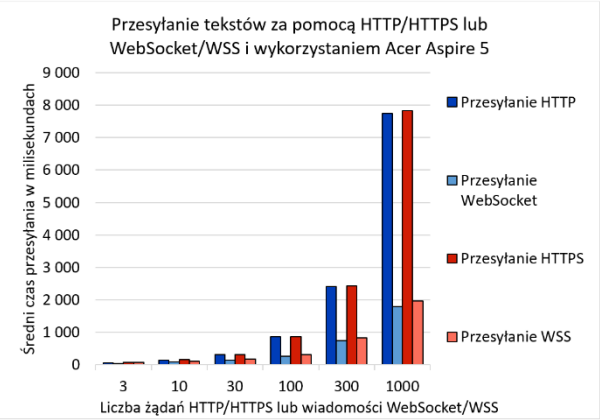
W pierwszym kroku zmierzono czas przesyłania tekstów pomiędzy starszym, wolniejszym komputerem klienckim a serwerem za pomocą protokołu HTTP i HTTPS oraz WebSocket i WebSocket Secure

w określonej ilości żądań. Na podstawie wyników wykonano wykres słupkowy ze średnich wartości, który przedstawiono na rysunku numer 13.



Rysunek 13: Średnie czasy przesyłania tekstów pomiędzy komputerem klienckim Asus K50IN a serwerem za pomocą protokołów HTTP/HTTPS i WebSocket/WSS.

W drugim kroku zmierzono czas przesyłania tekstów pomiędzy nowszym, szybszym komputerem klienckim a serwerem za pomocą protokołu HTTP i HTTPS oraz WebSocket i WebSocket Secure w określonej ilości żądań. Na podstawie wyników wykonano wykres słupkowy ze średnich wartości, który przedstawiono na rysunku numer 14.



Rysunek 14: Średnie czasy przesyłania tekstów pomiędzy komputerem klienckim Acer Aspire 5 a serwerem za pomocą protokołów HTTP/HTTPS i WebSocket/WSS

Najważniejszym faktem do stwierdzenia na powyższych wykresach jest to, że zastosowanie szyfrowania TLS dla protokołu HTTP oraz WebSocket, ma minimalnie negatywny wpływ na czas w wysyłaniu danych z komputerów klienckich na serwer i ich pobieraniu z serwera poprzez klientów, bez względu na szybkość komputera klienckiego. Przyczyną takiego zjawiska są niskie wymagania sprzętowe w szyfrowaniu i deszyfrowaniu danych.

5. Wnioski

Z przeprowadzonych badań wynika, że wykorzystując protokół WebSocket, w porównaniu do HTTP, można osiągnąć wzrost wydajności o kilkaset procent. Można nawet przewidywać, że za jego pomocą przesłanie kilku lub jednej kopii takich danych (w porównaniu do zwykłego protokołu HTTP) może być nieco szybsze. Wzrost ten jest bardziej zauważalny dla transmisji powyżej 100 kopii danych, dlatego zastosowanie tej nowej technologii świetnie sprawdzi się w aplikacjach wymagających przesyłanie setek, a nawet tysięcy porcji danych na sekundę. Autor nadmienił, że badania zostały wykonane w pełni domyślnych ustawieniach protokołów, dlatego jeśli zwiększy się narzuty żądania lub odpowiedzi HTTP, różnica będzie jeszcze większa, co również udowodniono. Zauważono również, że protokół WebSocket tak samo wydajnie pracował na wolniejszym, jak i szybszym laptopie, natomiast wykonywanie żądań HTTP i przetwarzanie jego odpowiedzi znacznie lepiej przebiegało na nowszym komputerze klienckim. Zastosowanie szyfrowania TLS, zarówno dla HTTP, jak i WebSocket ma znikomy wpływ na obniżenie szybkości tych protokołów.

Ostatecznie, można wysunąć wniosek, że nie ma sensu implementować technologii WebSocket dla przesłania kilku lub kilkunastu porcji danych, gdyż w tym wypadku wystarczy wykorzystać zwykłe odpytywanie HTTP – tym bardziej, że powstają coraz szybsze komputery, które coraz sprawniej obsłużą tą technikę, pomijając sam fakt nadchodzącego szybkiego Internetu, który niweluje nadmiarowość i opóźnienia generowane przez to rozwiązanie.

Literatura

[1] World Wide Web, w: Wikipedia, The Free Encyclopedia, https://en.wikipedia.org/wiki/World_Wide_Web, 13.01.2021.

[2] Hypertext Transfer Protocol, w: Wikipedia, The Free Encyclopedia, https://en.wikipedia.org/wiki/Hypertext_Transfer_Protocol, [13.01.2021].

[3] WebSockets – A Conceptual Deep-Dive, w: Ably Realtime, https://www.ably.io/concepts/websockets, [13.01.2021].

[4] Ajax (programming), w: Wikipedia, The Free Encyclopedia, https://en.wikipedia.org/wiki/Ajax_(programming), [13.01.2021].

[5] WebSocket Simplified, w: Coding Simplified With Shad, https://iamshadmirza.hashnode.dev/websocket-simplified-cjxjzcu0m002i3hs1eewt2p80, [13.01.2021].

[6] WebSocket, w: Wikipedia, The Free Encyclopedia, https://en.wikipedia.org/wiki/WebSocket, [13.01.2021].

[7] RFC 6455 – The WebSocket Protocol, w: IETF Tools, https://tools.ietf.org/html/rfc6455, [13.01.2021].

[8] Real-time web, w Wikipedia, The Free Encyclopedia, https://en.wikipedia.org/wiki/Real-time_web, 13.01.2021.

[9] W. Słodziak, Z. Nowak: Performance Analysis of Web Systems Based on XMLHttpRequest, Server-Sent Events and WebSocket. Springer International Publishing, 2016.

[10] Benchmark 5-milion Websockets, w: Oat++, https://oatpp.io/benchmark/websocket/5-million/, [13.01.2021].