Buduj wydajne aplikacje internetowe!

Przewodnik

Wydajne aplikacje internetowe

HELION

O'REILLY®

Ilya Grigorik

Tytuł oryginału: High Performance Browser Networking

Tłumaczenie: Andrzej Watrak (wstęp, rozdz. 9 – 18), Lech Lachowski (rozdz. 1 – 8)

ISBN: 978-83-246-8894-4

Wydawnictwo HELION
ul. Kościuszki 1c, 44-100 GLIWICE
tel. 32 231 22 19, 32 230 98 63
e-mail: helion@helion.pl
WWW: http://helion.pl (księgarnia internetowa, katalog książek)

Drogi Czytelniku!
Jeżeli chcesz ocenić tę książkę, zajrzyj pod adres
http://helion.pl/user/opinie/wydapi
Możesz tam wpisać swoje uwagi, spostrzeżenia, recenzję.

Printed in Poland.

Spis treści

ROZDZIAŁ 12.

Protokół HTTP 2.0

Protokół HTTP 2.0 sprawia, że nasza aplikacja jest szybsza, prostsza i bardziej spójna (to rzadkie połączenie cech). Dzięki niemu można usunąć z aplikacji wiele tymczasowych korekt protokołu HTTP 1.1, ponieważ teraz zostały one wprowadzone w warstwie transportowej. Co więcej, nowa wersja otwiera mnóstwo zupełnie nowych możliwości optymalizacji aplikacji i poprawy wydajności!

Podstawowym celem protokołu HTTP 2.0 jest zmniejszenie opóźnień aplikacji poprzez udostępnienie pełnej multipleksacji żądań i odpowiedzi, zminimalizowanie narzutu protokołu za pomocą skutecznej kompresji pól nagłówka HTTP, nadawanie żądaniom priorytetów i wypychanie zasobów. Aby zaimplementować te funkcjonalności, wprowadzono wiele dodatkowych ulepszeń, takich jak sterowanie przepływem, obsługa błędów i mechanizmy aktualizacji danych. Są to najważniejsze funkcjonalności, które powinien rozumieć i wykorzystywać w swoich aplikacjach każdy programista stron WWW.

Wersja HTTP 2.0 w żaden sposób nie zmienia sposobu wykorzystania protokołu HTTP przez aplikację. Wszystkie podstawowe koncepcje, takie jak metody HTTP, kody stanów, identyfikatory URI i pola nagłówka, wciąż istnieją. Natomiast wersja HTTP 2.0 zmienia sposób formatowania (ramkowania) danych i przesyłania ich między klientem a serwerem. Obie strony zarządzają procesem komunikacji, a wszystkie jego zawiłości są ukryte przed aplikacją w nowej warstwie ramkowania. W rezultacie wszystkie istniejące aplikacje mogą być stosowane bez modyfikacji. To są dobre wiadomości.

Jednak my jesteśmy zainteresowani nie tylko samym dostarczaniem działającej aplikacji. Naszym celem jest osiągnięcie jak najwyższej wydajności! Protokół HTTP 2.0 oferuje kilka nowych metod optymalizacji, które wcześniej nie były możliwe, a teraz można zastosować je w naszej aplikacji. Naszym zadaniem jest jak najlepiej je wykorzystać. Przyjrzyjmy się bliżej, co jest w środku protokołu.

Standard w trakcie tworzenia

Obecnie protokół HTTP 2.0 jest na etapie aktywnego tworzenia. Podstawowe konstrukcje architektury, zasady działania i nowe funkcjonalności są jasno określone, ale tego samego nie można powiedzieć o konkretnych niskopoziomowych szczegółach implementacyjnych. Z tego powodu skupimy się na architekturze protokołu i jej implementacji, a format przesyłanych danych będzie omówiony bardzo ogólnie, w stopniu wystarczającym do zrozumienia działania protokołu i wynikających z niego konsekwencji.

Aby poznać najnowszą specyfikację i status standardu protokołu HTTP 2.0, odwiedź stronę IETF pod adresem *http://tools.ietf.org/html/draft-ietf-httpbis-http2*.

Historia protokołu i jego związek ze SPDY

SPDY jest eksperymentalnym protokołem opracowanym przez Google, udostępnionym w połowie roku 2009. Jego podstawowym celem było skrócenie czasu ładowania stron WWW i rozwiązanie kilku znanych ograniczeń wydajnościowych protokołu HTTP 1.1. W szczególności projekt miał nakreślone następujące cele:

- skrócenie czasu ładowania strony o 50%,
- uniknięcie konieczności zmiany zawartości stron WWW przez ich twórców,
- zminimalizowanie złożoności wdrożenia, uniknięcie zmian w infrastrukturze sieciowej,
- opracowanie nowego protokołu wspólnie ze społecznością tworzącą otwarty kod,
- zebranie rzeczywistych danych wydajnościowych w celu zatwierdzenia lub odrzucenia eksperymentalnego protokołu.

Aby osiągnąć poprawę czasu ładowania strony o 50%, protokół SPDY w założeniu miał efektywniej wykorzystywać połączenia TCP przez wprowadzenie nowej warstwy ramkowania binarnego, umożliwiającej multipleksację żądań i odpowiedzi, nadawanie priorytetów i minimalizację lub eliminację niepotrzebnego opóźnienia sieciowego (patrz podrozdział „Opóźnienie sieciowe, wąskie gardło w wydajności" w rozdziale 10.).

Niedługo po pierwszej zapowiedzi protokołu, inżynierowie Google, Mike Belshe i Roberto Peon, ogłosili swoje pierwsze wyniki, dokumentację i kod źródłowy eksperymentalnej implementacji nowego protokołu SPDY:

> Jak dotąd, przetestowaliśmy protokół SPDY jedynie w warunkach laboratoryjnych. Wyniki wyglądają bardzo obiecująco. Podczas otwierania 25 najpopularniejszych stron WWW poprzez symulowane domowe połączenie sieciowe stwierdziliśmy znaczną poprawę wydajności — strony ładowały się o 55% szybciej.
>
> — *A 2x Faster Web* („2 × szybsza sieć")
>
> Chromium Blog

Przeskoczmy szybko kilka lat wprzód, do roku 2012. Nowy, eksperymentalny protokół jest obsługiwany przez przeglądarki Chrome, Firefox, Opera, a wiele dużych stron (np. Google, Twitter, Facebook) oferuje protokół SPDY kompatybilnym z nim klientom. Innymi słowy, okazało się, że protokół SPDY zaoferował ogromne korzyści wydajnościowe, a wskutek jego coraz szerszego zastosowania stał się de facto standardem. W efekcie grupa HTTP Working Group (HTTPWG), wyciągając wnioski z lekcji protokołu SPDY, opublikowała na początku roku 2012 nowy protokół HTTP 2.0 i wprowadziła go jako oficjalny standard.

Droga do HTTP 2.0

Protokół SPDY był zalążkiem protokołu HTTP 2.0, ale nie jest to właściwy protokół HTTP 2.0. Pierwsza specyfikacja protokołu HTTP 2.0 pojawiła się na początku 2012 roku i po burzliwych dyskusjach wewnątrz grupy HTTPWG specyfikacja SPDY została przyjęta jako punkt startowy w pracy nad nowym standardem. Od tej pory do oficjalnego standardu protokołu HTTP 2.0 zostało wprowadzonych i będzie dalej wprowadzanych wiele zmian i ulepszeń.

Jednak zanim wybiegniemy za daleko, warto przejrzeć wstępną propozycję protokołu HTTP 2.0, ponieważ uwidacznia ona zakres i najważniejsze założenia protokołu:

> Oczekuje się, że protokół HTTP 2.0 wprowadzi następujące zmiany:
>
> - W przypadku większości stron w znaczący i wymierny sposób zostanie zmniejszone opóźnienie postrzegane przez użytkowników w porównaniu z protokołem HTTP 1.1 wykorzystującym TCP.
> - Zostanie rozwiązany problem blokowania początku kolejki w protokole HTTP.
> - Do zrównoleglenia obsługi żądań nie będzie wymagane nawiązywanie wielu połączeń z serwerem, dzięki czemu w większym stopniu będzie wykorzystany protokół TCP, szczególnie pod kątem sterowania spiętrzeniami ruchu.
> - Zostanie zachowane działanie protokołu HTTP 1.1 zgodnie z istniejącą dokumentacją, wykorzystujące metody HTTP (ale nie tylko), kody stanów, identyfikatory URI i pola nagłówka tam, gdzie będzie to zasadne.
> - Będzie jasno zdefiniowana interakcja z protokołem HTTP 1.x, szczególnie w urządzeniach pośrednich.
> - Będą jasno określone miejsca umożliwiające rozbudowę protokołu i zasady ich właściwego wykorzystania.
>
> Oczekuje się, że ostateczna specyfikacja spełni powyższe oczekiwania w przypadku istniejących wdrożeń protokołu HTTP, dotyczących szczególnie przeglądania internetu (za pomocą urządzeń stacjonarnych i mobilnych), użycia poza przeglądarkami (HTTP API), udostępniania stron WWW (w różnej skali wielkości) i obsługi w urządzeniach pośrednich (serwerach proxy, zaporach, odwrotnych serwerach proxy i w sieciach CDN). Analogicznie bieżące i przyszłe rozszerzenia protokołu HTTP/1.x (np. nagłówki, metody, kody stanu, dyrektywy pamięci podręcznej) będą obsługiwane w nowym protokole.
>
> — Statut grupy roboczej HTTPbis
>
> *HTTP 2.0*

Krótko mówiąc, protokół HTTP 2.0 miał na celu rozwiązanie wielu znanych ograniczeń wydajnościowych poprzednich standardów 1.x, jak również ich rozwinięcie, lecz nie zastąpienie. Wykorzystanie protokołu HTTP w aplikacji pozostało takie samo i nie zostały wprowadzone żadne zmiany do oferowanych funkcjonalności ani podstawowych koncepcji, takich jak metody, kody stanów, identyfikatory URI i pola nagłówka. Tego typu zmiany zostały jawnie wykluczone. Czy w takim razie oznaczenie „2.0” jest uzasadnione?

Powodem zwiększenia głównego numeru wersji do 2.0 jest zmiana sposobu wymiany danych między klientem a serwerem. Aby osiągnąć wytyczone cele wydajnościowe, protokół HTTP 2.0 wprowadza nową warstwę ramkowania binarnego, która nie jest kompatybilna wstecz z poprzednią wersją 1.x. Stąd numer 2.0.

O ile nie implementujesz własnego serwera WWW lub nietypowego klienta i nie operujesz na podstawowych gniazdach TCP, to może się okazać, że nawet nie zauważysz żadnej z faktycznych zmian wprowadzonych przez protokół HTTP 2.0. Całe nowe niskopoziomowe ramkowanie jest wykonywane za Ciebie przez przeglądarkę i serwer. Jedyną różnicą może być dostępność nowych i opcjonalnych funkcjonalności interfejsu API, na przykład wypychania zasobów!

Na koniec warto omówić prognozy rozwoju protokołu HTTP 2.0. Opracowanie nowej wersji protokołu realizującego całą komunikację w sieci WWW jest nietrywialnym zadaniem, wymagającym wielu starannych przemyśleń, eksperymentów i koordynacji prac. Dlatego prognozowanie rozwoju protokołu HTTP 2.0 jest niebezpiecznym procederem. Protokół będzie gotowy wtedy, gdy będzie gotowy. Oznacza to, że grupa HTTP-WG robi szybkie postępy i oficjalnie wyznaczyła następujące etapy:

- marzec 2012 — ogłoszenie zapotrzebowania na protokół HTTP 2.0,
- wrzesień 2012 — pierwsza specyfikacja protokołu HTTP 2.0,
- lipiec 2013 — pierwsza specyfikacja implementacji protokołu 2.0,
- kwiecień 2014 — ostatnie prace grupy Working Group nad protokołem HTTP 2.0,
- listopad 2014 — przesłanie do grupy IESG specyfikacji protokołu HTTP 2.0 jako propozycji nowego standardu.

Duża przerwa między rokiem 2012 a 2014 przypadła na intensywne prace edycyjne i eksperymentalne. W zależności od postępów i opinii ze strony administratorów i środowiska jako całości daty mogą się zmienić. Dobra wiadomość jest taka, że na koniec roku 2013 harmonogram był dotrzymany!

Wspólna ewolucja protokołów HTTP 2.0 i SPDY

Grupa pracująca nad protokołem HTTP przyjęła w lecie 2012 roku specyfikację protokołu SPDY v2 jako punkt startowy prac nad standardem HTTP 2.0. Jednak gdy to się stało, prace nad protokołem SPDY nie ustały. Wręcz przeciwnie, protokół SPDY równolegle ewoluował:

- W 2012 roku została ogłoszona wersja SPDY v3 ze zaktualizowanym formatem ramkowania i pierwszą implementacją sterowania przepływem.
- Na przełomie lat 2013 i 2014 została udostępniona wersja SPDY v4 ze zaktualizowanym (ponownie) formatem ramkowania, usprawnioną obsługą priorytetów, sterowaniem przepływem i zaimplementowanym wypychaniem zasobów.

Powód kontynuacji rozwoju protokołu SPDY jest prosty — jest to motor napędowy w eksperymentach z nowymi funkcjonalnościami i propozycjami dla protokołu HTTP 2.0. To, co dobrze wygląda na papierze, może nie działać w praktyce i *vice versa*. Protokół SPDY jest drogą do testów i oceny każdej propozycji przed zawarciem jej w standardzie 2.0.

Ten stopniowy proces rozwoju i wspólnej ewolucji protokołów SPDY i HTTP 2.0 zadał wiele pracy programistom, ale w praktyce przyniesie również wiele korzyści — bardziej zwartą i dokładnie przetestowaną specyfikację, jak również implementację serwera i klienta, które także mogą równolegle wspólnie ewoluować. W rzeczywistości, gdy pojawi się protokół HTTP 2.0 z oznaczeniem „gotowy", będziemy mieć dokładnie przetestowane implementacje popularnych serwerów i klientów! W tym momencie protokół SPDY będzie mógł przejść na emeryturę, a na środku sceny pojawi się protokół HTTP 2.0.

Cele projektowe i techniczne

Protokół HTTP 1.x został zaprojektowany pod kątem prostoty implementacji. Protokół HTTP 0.9 był jednowierszową wersją, która zapoczątkowała rozwój sieci WWW. Protokół HTTP 1.0 usystematyzował rozszerzenia wersji 0.9 w nieformalnym standardzie, a wersja 1.1 stała się

oficjalnym standardem IETF (patrz rozdział 9.). W ten sposób protokół 0.9 – 1.x stał się tym, czym miał się stać — jednym z najbardziej rozpowszechnionych i najszerzej zaimplementowanych protokołów aplikacyjnych w internecie.

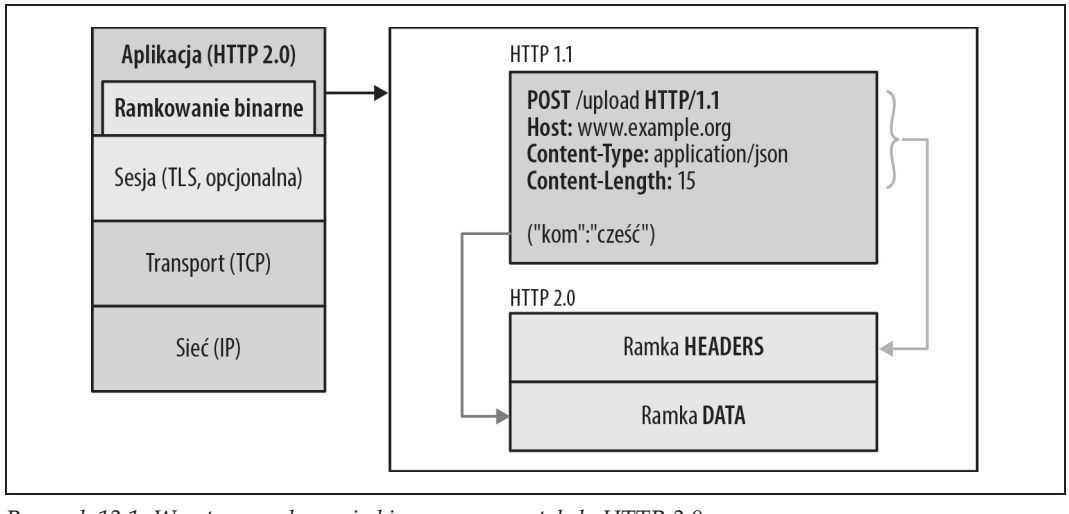
Niestety, prostota implementacji szła w parze z kosztami wydajności aplikacji, co stanowiło lukę, którą właśnie protokół HTTP 2.0 z założenia ma wypełnić:

> Enkapsulacja HTTP/2.0 umożliwia lepsze wykorzystanie zasobów sieciowych i dzięki kompresji pól nagłówka i jednoczesnemu przesyłaniu wielu komunikatów w ramach tego samego połączenia zmniejszenie postrzeganych przez użytkowników opóźnień. Protokół wprowadza również spontaniczne wypychanie zasobów z serwerów do klientów.
>
> — HTTP/2.0
>
> *Szkic 4.*

Prace nad protokołem HTTP 2.0 trwają, co oznacza, że szczegółowe instrukcje, jak kodować bity w każdej ramce, jakie będą nazwy poszczególnych pól i podobne niskopoziomowe szczegóły, mogą się zmienić. Jednak mimo tego, że instrukcje „jak" będą się zmieniać, podstawowe cele projektowe i techniczne, którym poświęcimy większą część naszego omówienia, zostały uzgodnione i są znane.

Warstwa ramkowania binarnego

Rdzeniem wszystkich udoskonaleń wydajnościowych wprowadzonych w protokole HTTP 2.0 jest nowa warstwa **ramkowania binarnego** (patrz rysunek 12.1), która określa sposób enkapsulacji i przesyłania komunikatów między klientem a serwerem.



Rysunek 12.1. Warstwa ramkowania binarnego w protokole HTTP 2.0

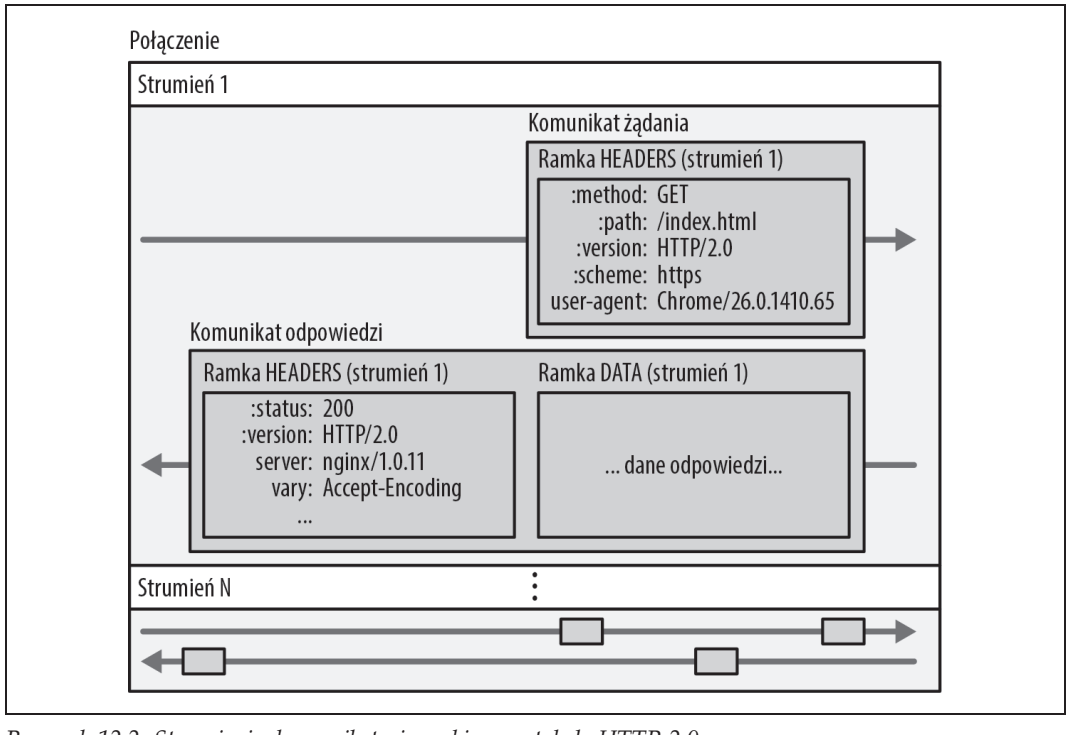
Termin „warstwa" określa nowy mechanizm wprowadzany pomiędzy interfejsem gniazda a interfejsem API wyższego poziomu udostępnianym naszej aplikacji. Elementy protokołu HTTP, takie jak słowa kluczowe, metody i nagłówki, pozostają takie same, ale zmiana polega na sposobie ich kodowania podczas transmisji. W odróżnieniu od protokołu HTTP 1.x, opartego na zwykłym tekście podzielonym znakami nowego wiersza, w protokole HTTP 2.0 wszystkie przesyłane dane są podzielone na mniejsze komunikaty i ramki zakodowane w formacie binarnym.

W rezultacie zarówno klient, jak i serwer, aby mogli się wzajemnie rozumieć, muszą stosować nowy mechanizm kodowania binarnego. Klient używający protokołu HTTP 1.x nie zrozumie serwera używającego tylko protokołu 2.0, i odwrotnie. Na szczęście nasza aplikacja pozostanie w błogiej nieświadomości istnienia tych wszystkich zmian, ponieważ to klient i serwer wykonają za nas całe niezbędne ramkowanie.

Protokół HTTPS jest kolejnym doskonałym przykładem zastosowania ramkowania binarnego — wszystkie komunikaty HTTP są kodowane i dekodowane w sposób niewidoczny dla nas (patrz podrozdział „Protokół TLS rekordu" w rozdziale 4.), dzięki czemu możliwa jest bezpieczna komunikacja między klientem a serwerem bez konieczności wprowadzania jakichkolwiek modyfikacji w aplikacji. Protokół HTTP 2.0 działa w podobny sposób.

Strumienie, komunikaty i ramki

Wprowadzenie nowego mechanizmu ramkowania binarnego zmienia sposób wymiany danych (patrz rysunek 12.2) między klientem a serwerem. Aby opisać ten proces, musimy wprowadzić kilka nowych terminów charakterystycznych dla protokołu HTTP 2.0:



Rysunek 12.2. Strumienie, komunikaty i ramki w protokole HTTP 2.0

Strumień
: Dwukierunkowy przepływ bajtów w ramach nawiązanego połączenia.

Komunikat
: Pełna sekwencja ramek tworzących logiczny komunikat.

Ramka

Najmniejsza jednostka danych w protokole HTTP 2.0, zawierająca nagłówek identyfikujący przynajmniej strumień, do którego ramka należy.

Cała komunikacja w protokole HTTP 2.0 odbywa się w ramach połączenia, które może zawierać dowolną liczbę dwukierunkowych strumieni. Z kolei strumienie przesyłają komunikaty składające się z jednej ramki lub kilku ramek, które mogą być wymieszane. Ramki są następnie składane w komunikaty na podstawie identyfikatora strumienia zawartego w nagłówku.

Wszystkie ramki w protokole HTTP 2.0 stosują kodowanie binarne, a dane nagłówka są kompresowane. Powyższy rysunek ilustruje zależność pomiędzy strumieniami, komunikatami i ramkami, ale bez kodowania podczas przesyłania ich przez łącze — te szczegóły są opisane w podrozdziale „Krótkie wprowadzenie do ramkowania binarnego", dalej w tym rozdziale.

W tych kilku zwięzłych zdaniach jest upakowanych mnóstwo informacji. Przejrzyjmy je jeszcze raz. Znajomość terminologii strumieni, komunikatów i ramek stanowi podstawę zrozumienia protokołu HTTP 2.0:

- Cała *komunikacja* odbywa się w ramach jednego połączenia TCP.
- *Strumień* jest to wirtualny kanał wewnątrz połączenia, przenoszący komunikaty w obu kierunkach. Każdy strumień posiada unikatowy identyfikator (1, 2, … N).
- *Komunikat* jest logicznym komunikatem HTTP, takim jak żądanie lub odpowiedź, składającym się z jednej ramki lub wielu ramek.
- *Ramka* jest najmniejszą jednostką komunikacyjną, zawierającą określone informacje, np. nagłówek HTTP, dane itp.

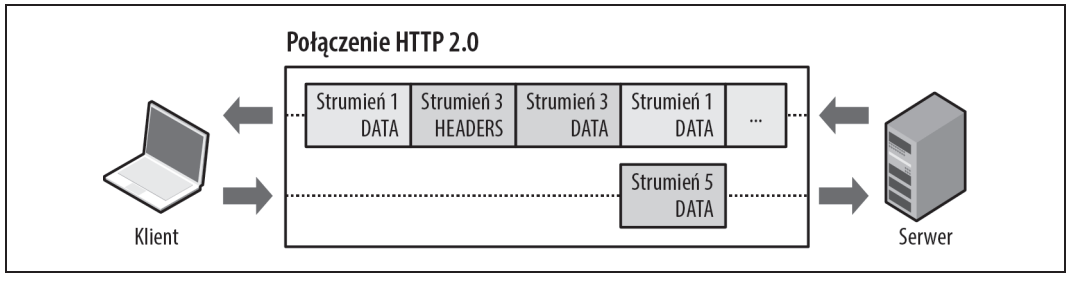
Krótko mówiąc, protokół HTTP 2.0 dzieli komunikację na poszczególne małe ramki tworzące komunikaty w ramach strumieni. Z kolei kilka strumieni może przesyłać równolegle komunikaty w ramach jednego połączenia TCP.

Multipleksacja żądań i odpowiedzi

Aby klient używający protokołu HTTP 1.x mógł zwiększyć wydajność, wysyłając równolegle kilka żądań, musiał użyć kilku połączeń TCP — patrz podrozdział „Użycie wielu połączeń TCP" w rozdziale 11. Takie działanie jest bezpośrednią konsekwencją modelu transmisji danych w protokole HTTP 1.x, w którym tylko jedna odpowiedź mogła być przesyłana w danej chwili (kolejkowanie odpowiedzi) w danym połączeniu. Co gorsza, ten sposób powodował blokowanie początku kolejki i nieefektywne wykorzystanie połączenia TCP.

Nowa warstwa ramkowania binarnego w protokole HTTP 2.0 usuwa te ograniczenia i umożliwia pełną multipleksację żądań i odpowiedzi, dzięki czemu klient i serwer mogą dzielić komunikaty HTTP na niezależne ramki (patrz rysunek 12.3), mieszać je i ponownie składać po drugiej stronie.

Rysunek 12.3 przedstawia kilka strumieni przesyłanych w ramach jednego połączenia. Klient wysyła do serwera ramkę DATA (strumień 5), a w tym czasie serwer wysyła do klienta przemieszaną sekwencję ramek w strumieniach 1 i 3. W rezultacie równolegle przesyłane są trzy żądania i odpowiedzi!



Rysunek 12.3. Multipleksacja żądań i odpowiedzi we współdzielonym połączeniu w protokole HTTP 2.0

Możliwość dzielenia komunikatów HTTP na niezależne ramki, mieszanie ich i składanie po drugiej stronie to najważniejsze udoskonalenie wprowadzane przez protokół HTTP 2.0. W rzeczywistości protokół oferuje wiele pochodnych korzyści wydajnościowych w całym stosie technologii WWW. Umożliwia między innymi:

- mieszanie wielu równoległych *żądań* bez ich wzajemnego blokowania,
- mieszanie wielu równoległych *odpowiedzi* bez ich wzajemnego blokowania,
- użycie *jednego połączenia* do równoległego przesyłania wielu żądań i odpowiedzi,
- szybsze ładowanie stron dzięki eliminacji niepotrzebnych opóźnień,
- usunięcie z kodu aplikacji niepotrzebnych napraw usterek protokołu HTTP 1.x,
- i wiele innych funkcjonalności…

Nowa warstwa ramkowania binarnego w protokole HTTP 2.0 rozwiązuje problem blokowania początku kolejki występujący w protokole HTTP 1.1 i eliminuje konieczność stosowania wielu połączeń do równoległego przesyłania i przetwarzania żądań i odpowiedzi. Dzięki temu nasze aplikacje mogą być szybsze, prostsze i tańsze we wdrożeniu.

Multipleksacja żądań i odpowiedzi umożliwia wyeliminowanie wielu napraw usterek protokołu HTTP 1.x, takich jak konkatenacja plików, kompozycje obrazów i rozdrobnienie domeny (patrz podrozdział „Optymalizacja protokołu HTTP 1.x” w rozdziale 13.). Ponadto dzięki zmniejszeniu liczby wymaganych połączeń TCP w protokole HTTP 2.0 zmniejsza się również koszt procesorów i pamięci zarówno po stronie klienta, jak i serwera.

Priorytety żądań

Po podzieleniu komunikatu HTTP na kilka osobnych ramek ich kolejność przesyłania może być zoptymalizowana i w ten sposób może być zwiększona wydajność naszej aplikacji. Aby to osiągnąć, każdemu strumieniowi jest przypisana 31-bitowa wartość oznaczająca priorytet:

- 0 oznacza strumień o najwyższym priorytecie,
- 2^{31}–1 oznacza strumienie o niższych priorytetach.

Dzięki priorytetom klient i serwer mogą stosować różne strategie przetwarzania w optymalnej kolejności poszczególnych strumieni, komunikatów i ramek. Serwer może określać priorytet przetwarzanego strumienia i kontrolować wykorzystanie zasobów (procesora, pamięci, pasma), a gdy będą dostępne dane odpowiedzi, określać priorytet wysyłania ważnych ramek do klienta.

Priorytety żądań przeglądarki i protokół HTTP 2.0

Podczas wyświetlania strony przez przeglądarkę nie wszystkie zasoby mają taki sam priorytet. Sam dokument HTML ma krytyczne znaczenie podczas tworzenia obiektu DOM. Plik CSS jest wymagany do utworzenia obiektu CSSOM. Tworzenie obiektów DOM i CSSOM może być blokowane przez zasoby JavaScript (patrz ramka „Obiekty DOM, CSSOM i JavaScript" w rozdziale 10.). Natomiast pozostałe zasoby, takie jak obrazy, są często pobierane z niższym priorytetem.

Aby skrócić czas ładowania strony, wszystkie nowoczesne przeglądarki nadają priorytety żądaniom na podstawie rodzaju zasobu, jego położenia na stronie, a nawet priorytetu wyuczonego podczas ostatniego otwarcia. Jeżeli na przykład podczas poprzedniego otwarcia strona była blokowana przez jakiś zasób, to w przyszłości może być mu przypisany wyższy priorytet.

W protokole HTTP 1.x przeglądarka miała ograniczone możliwości ustanawiania priorytetów danych. Protokół nie obsługiwał multipleksacji i nie sposób było przekazać serwerowi priorytetu żądania. Zamiast tego stosowane były jednoczesne połączenia, które umożliwiały ograniczone zrównoleglenie transmisji do sześciu żądań na serwer. W rezultacie żądania czekały na dostępne połączenie w kolejce po stronie klienta, co wprowadzało niepotrzebne opóźnienie sieciowe. Teoretycznie kolejkowane żądań HTTP, opisane w rozdziale 11., było próbą częściowego rozwiązania tego problemu, ale w praktyce nie zostało zastosowane.

Protokół HTTP 2.0 naprawia wszystkie te niedoskonałości. Przeglądarka może wysyłać żądanie każdego zasobu natychmiast po jego wykryciu, określać priorytet każdego strumienia i pozwolić serwerowi na określenie optymalnej strategii wysłania odpowiedzi. W ten sposób wyeliminowane jest niepotrzebne opóźnienie związane z kolejkowaniem żądań i możliwe jest osiągnięcie najefektywniejszego wykorzystania każdego połączenia.

Protokół HTTP 2.0 nie określa żadnego konkretnego algorytmu obsługi priorytetów, oferuje jedynie mechanizm wymiany priorytetowych danych pomiędzy klientem a serwerem. Zatem priorytety są mi, a strategia ich obsługi może być różna, w zależności od implementacji protokołu po stronie klienta i serwera. Klient powinien dostarczać prawidłowe informacje o priorytetach danych, a serwer powinien dostosować ich przetwarzanie i dostarczanie zgodnie ze wskazanymi priorytetami strumieni.

Nie masz wpływu na wiarygodność priorytetów nadawanych przez klienta, możesz jednak mieć wpływ na serwer. Dlatego starannie wybieraj serwer obsługujący protokół HTTP 2.0! Dla zobrazowania tego zagadnienia rozważmy następujące pytania:

- Co się stanie, jeżeli serwer zlekceważy wszystkie informacje o priorytetach?
- Czy wszystkie strumienie o wysokim priorytecie muszą mieć pierwszeństwo?
- Czy wystąpią przypadki, w których strumienie o różnych priorytetach będą musiały być wymieszane?

Jeżeli serwer zlekceważy informacje o priorytetach, to może w niezamierzony sposób spowolnić działanie aplikacji, na przykład zablokować wyświetlanie strony w przeglądarce, wysyłając do niej obrazy, gdy tymczasem strona może oczekiwać na krytyczne pliki CSS i JavaScript. Z drugiej strony narzucanie ściśle określonych priorytetów również może prowadzić do nieoptymalnych scenariuszy, na przykład ponownego zablokowania początku kolejki, gdy jedno długotrwałe żądanie niepotrzebnie wstrzyma wysyłanie innych zasobów.

Ramki o różnych priorytetach mogą i powinny być mieszane na serwerze. Tam, gdzie to możliwe, strumienie o wysokim priorytecie powinny mieć pierwszeństwo, zarówno w kwestii kolejności przetwarzania, jak i przydzielania pasma transmisyjnego pomiędzy klientem a serwerem. Niemniej jednak, aby jak najlepiej wykorzystać połączenie, wymagane jest zróżnicowanie priorytetów.

Jedno połączenie na serwer

Dzięki nowemu mechanizmowi ramkowania protokół HTTP 2.0 nie potrzebuje wielu połączeń TCP, aby równolegle multipleksować wiele strumieni. Zamiast tego każdy strumień jest dzielony na wiele ramek, które mogą być mieszane i przesyłane z określonymi priorytetami. W rezultacie wszystkie połączenia są trwałe, a pomiędzy klientem a serwerem jest używane tylko jedno połączenie.

> Na podstawie pomiarów w laboratorium dzięki zastosowaniu mniejszej liczby połączeń od klienta stwierdziliśmy zdecydowane zmniejszenie opóźnień. Całkowita liczba pakietów przesyłanych za pomocą protokołu HTTP 2.0 została zmniejszona aż o 40% w stosunku do poprzednich wersji protokołu. Obsługa przez serwer dużej liczby połączeń również zaczynała stanowić problem ze skalowalnością systemu, a protokół HTTP 2.0 zmniejszył to obciążenie.
>
> — HTTP/2.0
> *Szkic 2.*

Jedno połączenie na serwer znacznie zmniejsza wprowadzany przez nie narzut — zarządzania wymaga mniejsza liczba gniazd na całej ścieżce połączenia, zajmowana jest mniejsza pamięć, a połączenie ma większą przepływność. Do tego dochodzi wiele innych korzyści we wszystkich warstwach stosu:

- Spójna obsługa priorytetów różnych strumieni.
- Lepsza kompresja danych dzięki zastosowaniu jednego kontekstu kompresji.
- Zmniejszenie spiętrzeń ruchu w sieci dzięki mniejszej liczbie połączeń TCP.
- Krótszy okres wolnego startu i szybsza obsługa spiętrzeń ruchu i strat pakietów.

W większości przypadków transmisje danych w protokole są krótkie i gwałtowne, gdy tymczasem protokół TCP jest zoptymalizowany pod kątem długotrwałych, intensywnych przesyłów danych. Dzięki wykorzystaniu tego samego połączenia przez wszystkie strumienie protokół HTTP 2.0 może efektywniej wykorzystać połączenie TCP.

Przejście na protokół HTTP 2.0 nie tylko powinno zmniejszyć opóźnienia sieciowe, ale również zwiększyć przepływność i zmniejszyć koszty utrzymania infrastruktury!

Sterowanie przepływem

Multipleksacja wielu strumieni w ramach jednego połączenia TCP powoduje pojawienie się rywalizacji o współdzielone pasmo transmisyjne. Nadanie strumieniom priorytetów pomaga określić względną kolejność przesyłania danych, ale same priorytety nie wystarczą do sterowania podziałem zasobów między wiele strumieni lub połączeń. Aby rozwiązać ten problem, protokół HTTP 2.0 oferuje prosty mechanizm sterowania przepływem wewnątrz strumienia i połączenia.

Straty pakietów, łącza o wysokim opóźnieniu i wydajność protokołu HTTP 2.0

Zaraz, zaraz… Wymieniliśmy zalety stosowania jednego połączenia TCP na serwer, ale czy nie powoduje to jakichś potencjalnych problemów? Owszem, tak właśnie jest.

- Wyeliminowaliśmy blokowanie początku kolejki na poziomie protokołu HTTP, ale kolejka wciąż jest blokowana na poziomie protokołu TCP (patrz podrozdział „Blokowanie typu HOL" w rozdziale 2.).
- Zgodnie z regułą iloczynu przepływności i opóźnienia przepływność połączenia może być ograniczona, jeżeli skalowanie okna TCP będzie wyłączone.
- Jeżeli zostanie utracony pakiet, rozmiar okna protokołu TCP zostanie zmniejszony (patrz podrozdział „Zapobieganie zatorom" w rozdziale 2.), wskutek czego zmniejszana jest przepływność całego połączenia.

Każde zjawisko wymienione w powyższej liście może mieć niekorzystny wpływ zarówno na przepływność, jak i opóźnienie połączenia w protokole HTTP 2.0. Jednak pomimo tych ograniczeń eksperymenty dowodzą, że zastosowanie pojedynczego połączenia jest najlepszym sposobem w strategii wdrożenia protokołu HTTP 2.0:

> Dotychczasowe testy pokazały, że zalety kompresji i priorytetów przeważają nad niekorzystnym efektem blokowania początku kolejki (szczególnie w przypadku występowania strat pakietów).
>
> — HTTP/2.0 *Szkic 2.*

Podobnie jak w każdym procesie optymalizacyjnym, usunięcie jednego słabego punktu w wydajności powoduje pojawienie się innego. W przypadku protokołu HTTP 2.0 słabym punktem jest protokół TCP. Dlatego właśnie przypomnijmy jeszcze raz, że dobrze dostrojony stos TCP na serwerze ma krytyczny wpływ na wydajność protokołu HTTP 2.0.

Wciąż trwają badania mające na celu usunięcie powyższych ograniczeń i zwiększenie wydajności protokołu TCP w ogólności. Powstały rozszerzenia TCP Fast Open, Proportional Rate Reduction, powiększono początkowy rozmiar okna i wprowadzono inne zmiany. Trzeba przyznać, że w ten sposób protokół HTTP 2.0, w odróżnieniu od poprzednich wersji, nie musi opierać się na protokole TCP. Zastosowanie innych protokołów transportowych, na przykład UDP, nie jest wykluczone.

- Sterowanie przepływem realizowane jest w poszczególnych odcinkach, a nie w całym połączeniu.
- Sterowanie przepływem opiera się na ramkach `WINDOW_UPDATE`: odbiorca ogłasza, ile bajtów może odebrać ze strumienia i całego połączenia.
- Wielkość okna do sterowania przepływem jest aktualizowana przez ramkę `WINDOW_UPDATE` zawierającą identyfikator strumienia i wartość, o którą okno ma być powiększone.
- Sterowanie przepływem ma określony kierunek. Odbiorca może wybrać dowolną wielkość okna, wymaganą dla każdego strumienia i całego połączenia.
- Sterowanie przepływem może być wyłączone przez odbiorcę, zarówno dla pojedynczego strumienia, jak i całego połączenia.

Po nawiązaniu połączenia w protokole HTTP 2.0 klient i serwer wymieniają ramki `SETTINGS`, określające wielkości okien do sterowania przepływem w obu kierunkach. Opcjonalnie każda strona może wyłączyć sterowanie przepływem w określonym strumieniu lub całym połączeniu.

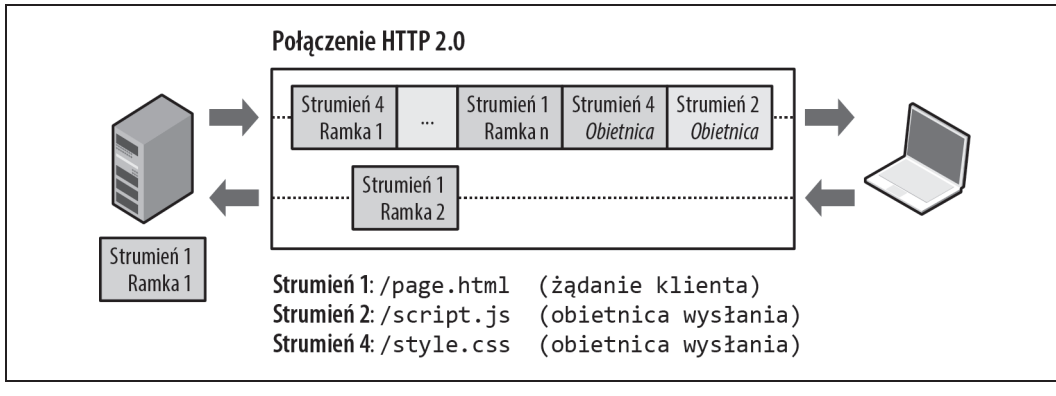
Czy powyższa lista nie przypomina Ci sterowania przepływem w protokole TCP? Na pewno, bo cały mechanizm jest identyczny — patrz podrozdział „Sterowanie przepływem" w rozdziale 2. Jednak samo sterowanie przepływem w protokole TCP jest niewystarczające, ponieważ nie są w nim rozróżniane strumienie w ramach połączenia HTTP 2.0. Dlatego zostało wprowadzone sterowanie przepływem na poziomie protokołu HTTP 2.0.

Standard HTTP 2.0 nie określa żadnego konkretnego algorytmu, zawartości ani momentów wysyłania ramek `WINDOW_UPDATE`. Twórca oprogramowania może wybrać swój własny algorytm odpowiedni do jego przypadku i zapewniający osiągnięcie największej wydajności.

Oprócz priorytetów określających względną kolejność przesyłania danych sterowanie przepływem może kontrolować wielkość zasobów zajmowanych przez każdy strumień w ramach połączenia HTTP 2.0. Odbiorca może zmniejszyć rozmiar okna dla określonego strumienia i ograniczyć prędkość przesyłania danych!

Wypychanie zasobów

Bardzo użyteczną nową funkcjonalnością protokołu HTTP 2.0 jest możliwość wysyłania przez serwer wielu odpowiedzi na żądanie klienta. Oznacza to, że serwer w odpowiedzi na żądanie klienta może *wypchnąć* dodatkowe zasoby (patrz rysunek 12.4) bez konieczności żądania ich wprost przez klienta!



Rysunek 12.4. Serwer inicjuje nowe strumienie (obietnice) do przesyłania zasobów

Po nawiązaniu połączenia HTTP 2.0 klient i serwer wymieniają ramki `SETTINGS`, które mogą ograniczać maksymalną liczbę równoległych strumieni w obu kierunkach. W rezultacie klient może ograniczyć liczbę wypychanych strumieni lub, ustawiając ją na zero, całkowicie wyłączyć wypychanie zasobów.

Dlaczego taki mechanizm jest potrzebny? Typowa aplikacja WWW składa się z dziesiątków zasobów, a wszystkie są wykrywane przez klienta w wyniku analizy dokumentu dostarczo-

nego przez serwer. Dlaczego więc nie wyeliminować dodatkowego opóźnienia i nie zlecić serwerowi wysłania od razu dodatkowych zasobów do klienta? Serwer już wie, jakich zasobów potrzebuje klient. Na tym polega ich wypychanie. W praktyce, jeżeli pliki CSS, JavaScript lub inne są osadzone za pomocą identyfikatorów URI (patrz podrozdział „Osadzanie zasobów" w rozdziale 11.), ma miejsce właśnie wypychanie zasobów!

Ręcznie osadzając zasoby w dokumencie, w rzeczywistości wypychamy je do klienta bez oczekiwania, aż ich zażąda. Jedyna różnica wprowadzana przez protokół HTTP 2.0 polega na przesunięciu tej czynności z aplikacji do samego protokołu, co przynosi istotne korzyści:

- Wypchnięte zasoby mogą być zapisane w pamięci podręcznej klienta.
- Wypchnięte zasoby mogą być odrzucone przez klienta.
- Wypchnięte zasoby mogą być wielokrotnie wykorzystane na różnych stronach.
- Wypchnięte zasoby mogą mieć priorytety nadane przez serwer.

Wszystkie wypchnięte zasoby podlegają zasadzie tego samego źródła. Dlatego serwer nie może wypchnąć do klienta dowolnej treści z innego serwera. Serwer musi być upoważniony do wypychania danej treści.

W rezultacie mechanizm wypychania zasobów eliminuje w większości przypadków konieczność osadzania zasobów, które jest stosowane w protokole HTTP 1.x. Jedyny uzasadniony przypadek bezpośredniego osadzania zasobów ma miejsce wówczas, gdy jest wymagany tylko przez jedną stronę i nie wprowadza dużego narzutu związanego z kodowaniem (patrz podrozdział „Osadzanie zasobów" w rozdziale 11.). W każdym innym przypadku Twoja aplikacja powinna wykorzystywać wypychanie zasobów!

Ramka PUSH_PROMISE

Wszystkie wypychane strumienie są inicjowane za pomocą ramki `PUSH_PROMISE` (obietnica wypchnięcia zasobu), która sygnalizuje zamiar wypchnięcia przez serwer opisanego w niej zasobu oprócz odpowiedzi na oryginalne żądanie klienta. Ramki `PUSH_PROMISE` zawierają tylko nagłówki HTTP obiecywanego zasobu.

Klient po otrzymaniu ramki `PUSH_PROMISE` ma możliwość odrzucenia strumienia (np. gdy zasób znajduje się już w pamięci podręcznej), co stanowi istotne udoskonalenie w stosunku do wersji HTTP 1.x. Osadzanie zasobów, będące popularną metodą „optymalizacji" w protokole HTTP 1.x, jest równoznaczne z ich „wciskaniem" — klient nie może ich odrzucić, jak również nie może umieszczać ich osobno w pamięci podręcznej.

Na koniec kilka ograniczeń funkcjonalności wypychania zasobów. Po pierwsze, serwer musi stosować mechanizm żądanie-odpowiedź i wypychać zasoby tylko w odpowiedzi na żądanie klienta. Serwer nie może sam zainicjować wypychanego strumienia. Po drugie, ramki `PUSH_PROMISE` muszą zostać wysłane przed wysłaniem odpowiedzi, aby uniknąć efektu wyścigu, tj. żądania przez klienta tych samych zasobów, które serwer zamierza wypchnąć.

Implementacja wypychania zasobów w protokole HTTP 2.0

Wypychanie zasobów otwiera wiele nowych możliwości optymalizacji transmisji danych w aplikacjach. W jaki sposób jednak serwer określa zasoby, które mogą lub muszą być wysłane?

Podobnie jak w przypadku priorytetów, standard HTTP 2.0 nie określa żadnego konkretnego algorytmu i decyzję pozostawia twórcom oprogramowania. Istnieje zatem wiele możliwych strategii, z których każda może być dostosowana do danej aplikacji lub serwera:

- Aplikacja w swoim kodzie może jawnie zainicjować wypychanie zasobów. Ten sposób wymaga ścisłej współpracy z serwerem obsługującym protokół HTTP 2.0, ale zapewnia programiście pełną kontrolę nad procesem.
- Aplikacja może zasygnalizować serwerowi w dodatkowym nagłówku HTTP, które zasoby powinny zostać wypchnięte. W ten sposób aplikacja jest oddzielona od interfejsu API serwera HTTP 2.0. Na przykład moduł `mod_spdy` serwera Apache sprawdza nagłówek `X-Associated-Content` zawierający listę zasobów, które powinny być wypchnięte.
- Serwer może automatycznie określić powiązane ze stroną zasoby bez udziału aplikacji. Może przeanalizować dokument i określić zasoby do wypchnięcia albo przeanalizować odebrany ruch i podjąć odpowiednie decyzje, np. określić zależności między zasobami na podstawie nagłówka `Referrer`, a następnie automatycznie wypchnąć krytyczne zasoby do klienta.

Powyższa lista strategii nie jest pełna, ale ilustruje szeroki zakres możliwości, od wykorzystania niskopoziomowego interfejsu API po w pełni zautomatyzowaną implementację. Można zapytać, czy serwer powinien za każdym razem wypychać te same zasoby, czy można zaimplementować lepszą strategię. Serwer może być inteligentny i na podstawie własnego modelu działania, ciasteczek lub innego mechanizmu próbować określić, które zasoby znajdują się w pamięci podręcznej, i podejmować odpowiednie czynności. Krótko mówiąc, wypychanie zasobów otwiera mnóstwo nowych możliwości wprowadzania innowacji.

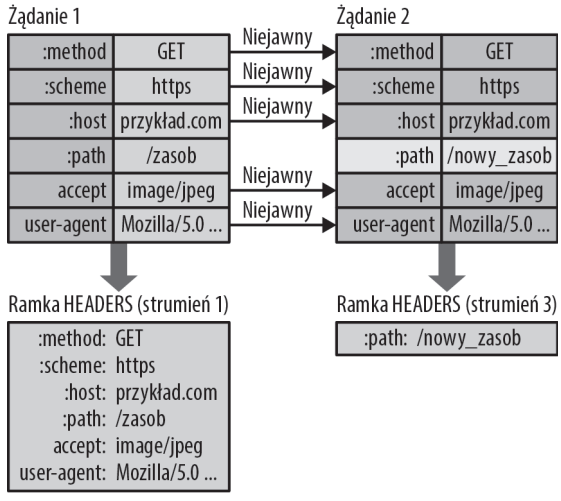
Na koniec warto zaznaczyć, że wypychane zasoby są umieszczane bezpośrednio w pamięci podręcznej, tak jak w przypadku żądania zainicjowanego przez klienta. Po stronie klienta nie ma żadnego interfejsu API ani funkcji zwrotnych JavaScript, które informowałyby o nadejściu wypchniętego zasobu. Cały mechanizm jest niewidoczny dla aplikacji uruchomionej w przeglądarce.

Kompresja nagłówka

Wszystkie przesyłane dane HTTP zawierają zestaw nagłówków opisujących przesyłane zasoby i ich właściwości. W protokole HTTP 1.x te metadane są zawsze przesyłane jako zwykły tekst i wprowadzają dodatkowy narzut na każde żądanie wielkości 500 – 800 bajtów albo kilku kilobajtów, jeżeli wymagane są ciasteczka HTTP (patrz podrozdział „Pomiar i kontrola narzutu protokołu” w rozdziale 11.). Aby zmniejszyć narzut i poprawić wydajność, protokół HTTP 2.0 kompresuje metadane w nagłówku:

- Zamiast ponownego przesyłania tych samych danych w każdym żądaniu i odpowiedzi protokół HTTP 2.0 stosuje „tabele nagłówków” po stronie klienta i serwera do śledzenia i przechowywania wcześniej przesłanych par klucz-wartość.
- Tabele nagłówków są tworzone dla całego połączenia HTTP 2.0 i są stopniowo aktualizowane zarówno po stronie klienta, jak i serwera.
- Każda nowa para klucz-wartość jest albo dołączana do istniejącej tabeli, albo zastępuje poprzednią parę.

W rezultacie obie strony połączenia HTTP 2.0 znają wcześniej przesłane nagłówki i ich wartości, co pozwala zakodować nowy zestaw nagłówków po prostu jako różnicę w stosunku do poprzedniego zestawu (patrz rysunek 12.5).



Rysunek 12.5. Kodowanie różnicowe nagłówków HTTP 2.0

Definicje pól nagłówka w protokole HTTP 2.0 pozostały niezmienione z kilkoma niewielkimi wyjątkami — wszystkie klucze są pisane małymi literami, a wiersz żądania jest podzielony na poszczególne pary klucz-wartość `:method`, `:scheme`, `:host` i `:path`.

Kompresja w protokołach SPDY, CRIME i HTTP 2.0

Pierwsze wersje protokołu SPDY do kompresji nagłówków HTTP stosowały metodę zlib z własnym słownikiem, co skutkowało zmniejszeniem przesyłanych danych nagłówków o 85 – 88% i znacznym skróceniem czasu ładowania strony:

> Na wąskim łączu DSL z prędkością transmisji w górę sieci równą 375 kb/s kompresja danych, w szczególności nagłówka żądania, prowadziła do znacznego skrócenia czasu ładowania niektórych stron (tj. tych, które wysyłały dużą liczbę żądań zasobów). Stwierdziliśmy skrócenie czasu ładowania strony o 45 – 1142 ms dzięki samej kompresji nagłówków.
>
> — *SPDY*
>
> chromium.org

Jednak latem 2012 roku został opublikowany opis ataku „CRIME" wykorzystującego luki w algorytmie kompresji TLS i SPDY i umożliwiającego przejęcie sesji. W rezultacie zrezygnowano z algorytmu kompresji zlib i zastąpiono go nowym, opisanym wcześniej algorytmem wykorzystującym tabele indeksów, który rozwiązywał problem bezpieczeństwa i w praktyce oferował porównywalną wydajność.

Pełny opis algorytmu kompresji w protokole HTTP 2.0 znajduje się pod adresem *http://tools.ietf.org/html/draft-ietf-httpbis-header-compression*.

W powyższym przykładzie drugie żądanie wymaga przesłania jedynie nagłówka `path`, który różni się od poprzedniego żądania. Wszystkie pozostałe nagłówki są pobierane z poprzedniego zestawu. W ten sposób protokół HTTP 2.0 zapobiega wielokrotnemu przesyłaniu tych samych danych, co znacznie zmniejsza narzut wprowadzany przez każde żądanie.

Powszechnie stosowane pary klucz-wartość (np. `user-agent`, `accept` itp.), które rzadko zmieniają się w trakcie trwania połączenia, wystarczy przesłać tylko raz. W praktyce, jeżeli w kolejnych żądaniach nagłówki nie zmieniają się (np. żądane są te same zasoby), narzut nagłówków jest równy zero bajtów. Wszystkie nagłówki są automatycznie pobierane z poprzedniego żądania!

Skuteczna aktualizacja i wykrywanie protokołu HTTP 2.0

Przełączenie na protokół HTTP 2.0 nie odbywa się w ciągu jednej nocy. Trzeba zaktualizować miliony serwerów, które będą stosować ramkowanie binarne, a miliardy klientów będą aktualizować swoje przeglądarki i biblioteki sieciowe.

Dobra wiadomość jest taka, że większość nowoczesnych przeglądarek zawiera skuteczny mechanizm aktualizacji, który umożliwia szybkie wykorzystanie protokołu HTTP 2.0 przy minimalnej interwencji większości użytkowników. Mimo tego jednak niektórzy użytkownicy mogą trwać przy starszych przeglądarkach, ale serwery i urządzenia pośrednie muszą zostać zaktualizowane do protokołu HTTP 2.0, co sprawia, że cały proces będzie dłuższy i wymagający dużego nakładu pracy i środków.

Protokół HTTP 1.x będzie jeszcze stosowany przez jakieś dziesięć lat, więc większość serwerów i klientów będzie musiała obsługiwać oba standardy — 1.x i 2.0. W rezultacie klient, który obsługuje protokół HTTP 2.0, będzie musiał wykryć podczas nawiązywania połączenia, czy serwer i wszystkie urządzenia pośrednie również obsługują ten protokół. Trzeba rozważyć trzy przypadki:

- inicjowanie nowego połączenia HTTPS za pomocą protokołu TLS negocjacji ALPN,
- inicjowanie nowego połączenia HTTP *z* wcześniejszym powiadomieniem,
- inicjowanie nowego połączenia HTTP *bez* wcześniejszego powiadomienia.

Do wykrywania i negocjowania protokołu HTTP 2.0, będącego częścią zwykłego połączenia HTTPS, jest stosowana negocjacja ALPN (ang. *Application Layer Protocol Negotiation*, negocjacja protokołu warstwy aplikacyjnej) — patrz podrozdziały „Procedura handshake TLS" i „Rozszerzenie ALPN" w rozdziale 4. Zmniejszenie opóźnienia sieciowego jest krytycznym celem protokołu HTTP 2.0 i dlatego podczas nawiązywania połączenia HTTPS jest zawsze stosowana negocjacja ALPN.

Nawiązanie połączenia HTTP 2.0 w zwykłym nieszyfrowanym kanale wymaga nieco więcej pracy. Ponieważ zarówno protokoły HTTP 1.0, jak i HTTP 2.0 działają na tym samym porcie (80), więc z powodu braku jakiejkolwiek dodatkowej informacji o obsłudze protokołu HTTP 2.0 przez serwer klient musi zastosować mechanizm *HTTP Upgrade* do wynegocjowania odpowiedniego protokołu:

```
GET /strona HTTP/1.1
Host: serwer.przyklad.com
Connection: Upgrade, HTTP2-Settings
Upgrade: HTTP/2.0 ❶
HTTP2-Settings: (dane SETTINGS) ❷
```

```
HTTP/1.1 200 OK ❸
Content-length: 243
Content-type: text/html

(... Odpowiedź HTTP 1.1 ...)
```

(lub)

```
HTTP/1.1 101 Switching Protocols ❹
Connection: Upgrade
Upgrade: HTTP/2.0

(...Odpowiedź HTTP 2.0 ...)
```

❶ — inicjujące żądanie HTTP 1.1 z nagłówkiem aktualizacji do protokołu HTTP 2.0.

❷ — kodowanie Base64 danych HTTP/2.0 SETTINGS.

❸ — serwer odmawia aktualizacji i zwraca odpowiedź za pomocą protokołu HTTP 1.1.

❹ — serwer akceptuje aktualizację i przełącza się na nowe ramkowanie.

W ten sposób w powyższej procedurze aktualizacji (`Upgrade`) serwer nieobsługujący protokołu HTTP 2.0 może natychmiast po odebraniu żądania zwrócić odpowiedź HTTP 1.1. Ewentualnie może potwierdzić aktualizację, zwracając odpowiedź `101 Switching Protocols` w formacie HTTP 1.1, a następnie natychmiast przełączyć się na protokół HTTP 2.0 i zwrócić odpowiedź, używając nowego ramkowania binarnego. W obu przypadkach wprowadzana jest dodatkowa wymiana komunikatów.

Aby klient i serwer mogli potwierdzić, że świadomie wybierają komunikację HTTP 2.0, obaj muszą również wysłać „nagłówek połączenia", będący ściśle określoną przez standard sekwencją bajtów. Ta wymiana służy jako mechanizm zapobiegający „fal-startowi" w przypadku klientów, serwerów i urządzeń pośrednich, które niekiedy akceptują aktualizację, nie rozumiejąc nowego protokołu. Taka wymiana nie wprowadza dodatkowych wymian komunikatów, jedynie kilka dodatkowych bajtów na początku połączenia.

Wdrożenie protokołu HTTP 2.0 z protokołem TLS i negocjacją ALPN

Informacja o tym, że serwer obsługuje protokół HTTP 2.0, nie gwarantuje, że za chwilę zostanie nawiązane niezawodne połączenie. Mówiąc prościej, protokół HTTP 2.0 musi być obsługiwany na całej ścieżce i jeżeli któreś z urządzeń pośrednich nie spełni tego warunku, połączenie może nie dojść do skutku.

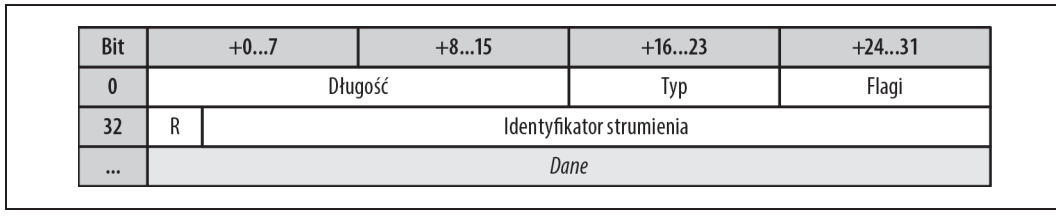
Dlatego mimo że protokół HTTP 2.0 nie wymaga zastosowania protokołu TLS, w praktyce jest to najlepsza metoda wdrożenia w przypadku dużej liczby urządzeń pośrednich (patrz ramka „Serwery proxy, urządzenia pośredniczące, TLS i nowe protokoły w sieci WWW" w rozdziale 4.). Aby uzyskać najlepsze efekty, każde wdrożenie powinno zaczynać się od protokołu TLS z negocjacją ALPN, jako uzupełnienie zwykłej procedury HTTP Upgrade.

Na koniec, jeżeli klient zechce, może zapamiętać lub pobrać informację o obsłudze protokołu HTTP 2.0 przez serwer innymi sposobami, np. z rekordu DNS, w wyniku ręcznej konfiguracji itp., i nie opierać się na procedurze `Upgrade`. Posiadając tę wiedzę, może zadecydować już na samym początku o wysłaniu przez niezaszyfrowany kanał ramek HTTP 2.0 i liczyć, że się uda nawiązać połączenie. W najgorszym przypadku połączenie nie zostanie nawiązane i klient będzie musiał wrócić do procedury `Upgrade` lub przełączyć się na tunel TLS z negocjacją ALPN.

Krótkie wprowadzenie do ramkowania binarnego

Najważniejszym ze wszystkich udoskonaleń protokołu HTTP 2.0 jest wprowadzenie nowej warstwy binarnego ramkowania z prefiksami długości. W porównaniu ze zwykłym tekstem rozdzielanym znakami nowego wiersza ramkowanie binarne oferuje bardziej zwartą reprezentację danych, które również można łatwiej i skuteczniej przetwarzać w kodzie.

Po nawiązaniu połączenia HTTP 2.0 klient i serwer komunikują się między sobą, wymieniając *ramki* będące najmniejszą jednostką danych w protokole. Wszystkie ramki mają taki sam 8-bajtowy nagłówek (patrz rysunek 12.6), zawierający długość ramki, jej typ, pole z flagami i 31-bitowy identyfikator strumienia.



Rysunek 12.6. Jednolity 8-bajtowy nagłówek ramki

- 16-bitowy prefiks `Długość` informuje, że jedna ramka może zawierać 2^{16}–1 bajtów danych, tj. około 64 kB, wyłączając 8-bajtowy nagłówek.
- 8-bitowe pole `Typ` określa sposób interpretacji pozostałej części ramki.
- 8-bitowe pole `Flagi` zawiera flagi komunikatów właściwe ramce danego typu.
- 1 bit jest zarezerwowany i zawsze ustawiony na 0.
- 31-bitowy identyfikator jednoznacznie określający strumień HTTP 2.0.

Niektórzy użytkownicy wolą do przeglądania ruchu HTTP 2.0 używać swojego ulubionego programu wyświetlającego dane w formacie heksagonalnym. Oprócz tego dostępne są wtyczki do programu Wireshark i podobne narzędzia prezentujące dane w bardziej zrozumiałej dla człowieka formie. Na przykład w przeglądarce Google Chrome po wpisaniu `chrome://internals#spdy` można obejrzeć wymieniane pakiety wyświetlone w formacie szesnastkowym.

Znając nagłówek ramki HTTP 2.0, możemy teraz napisać prosty program analizujący dowolny strumień bajtów HTTP 2.0 i identyfikować na podstawie pierwszych ośmiu bajtów każdej ramki jej typ, raportować informacje o flagach lub długościach. Ponadto, ponieważ każda ramka zawiera prefiks z długością, analizator może od razu szybko i skutecznie przeskakiwać na początek następnej ramki, co stanowi duże usprawnienie w stosunku do protokołu HTTP 1.1.

Gdy znany jest typ ramki, pozostała jej część może być przeanalizowana przez program. Standard protokołu HTTP 2.0 definiuje następujące typy ramek:

`DATA`	Używana do przesyłania treści komunikatów HTTP.
`HEADERS`	Używana do przekazywania dodatkowych pól nagłówka w strumieniu.
`PRIORITY`	Używana do przypisywania lub zmieniania priorytetów wskazywanego zasobu.
`RST_STREAM`	Używana do sygnalizowania nietypowego zakończenia strumienia.
`SETTINGS`	Używana do sygnalizowania danych konfiguracyjnych dotyczących sposobu komunikacji dwóch urządzeń po obu stronach połączenia.
`PUSH_PROMISE`	Używana do sygnalizowania obietnicy utworzenia strumienia i dostarczenia wskazanego zasobu.
`PING`	Używana do pomiaru czasu opóźnienia i przeprowadzenia testu „żywotności" połączenia.
`GOAWAY`	Używana do informowania drugiej strony, aby przerwała tworzenie strumienia w bieżącym połączeniu.
`WINDOW_UPDATE`	Używana do implementacji sterowania przepływem w strumieniu lub całym połączeniu.
`CONTINUATION`	Używana do kontynuacji sekwencji fragmentów bloku nagłówka.

Ramka `GOAWAY` umożliwia serwerowi wskazanie klientowi identyfikatora ostatnio przetwarzanego strumienia, dzięki czemu eliminowana jest seria żądań, a przeglądarka może inteligentnie wysłać żądanie jeszcze raz lub przerwać przetwarzanie bieżących żądań. Jest to ważna i potrzebna funkcjonalność umożliwiająca realizację bezpiecznej multipleksacji!

Ścisła implementacja powyższej terminologii ramek dotyczy w większości programistów tworzących oprogramowanie serwerów i klientów, którzy muszą troszczyć się o składnię każdego przepływu, obsługę błędów, przerywanie połączeń i wiele innych szczegółów. Dobra wiadomość jest taka, że wszystkie te zagadnienia są wyczerpująco opisane w oficjalnym standardzie. Jeżeli Cię one interesują, zapoznaj się z najnowszym szkicem specyfikacji.

Pola o stałej i zmiennej długości w protokole HTTP 2.0

Protokół HTTP 2.0 wykorzystuje wyłącznie pola o stałej długości. Narzut wprowadzany przez ramkę jest niewielki (8 bajtów nagłówka w ramce danych). Gdyby stosowane było kodowanie danych o zmiennej długości, wówczas oszczędności z tego tytułu nie kompensowałyby większego stopnia skomplikowania programu analizującego, jak również nie miałyby większego wpływu na wykorzystanie pasma i opóźnienie podczas wymiany danych.

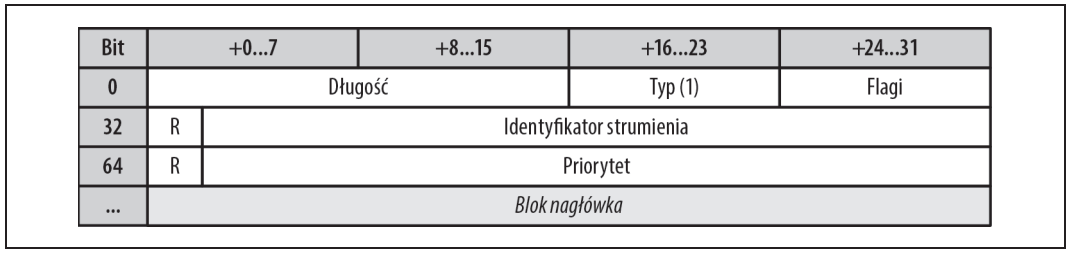
Gdyby kodowanie danych o zmiennej długości zmniejszyło narzut o 50%, to w przypadku pakietu sieciowego o długości 1400 bajtów, przesyłanego łączem o przepływności 1 Mb/s, umożliwiłoby zaoszczędzenie 4 bajtów (0,3%) i zmniejszenie opóźnienia każdej ramki o zaledwie 100 nanosekund.

Nawet jeżeli warstwa ramkowania jest ukryta przed naszą aplikacją, warto zrobić krok dalej i przyjrzeć się dwóm najczęściej spotykanym procedurom — inicjowania nowego strumienia i wymiany danych aplikacyjnych. Kierując się intuicją i zgadując, w jaki sposób żądanie lub odpowiedź są tłumaczone na poszczególne ramki, możemy odpowiedzieć na wiele pytań dotyczących wydajności protokołu HTTP 2.0.

Inicjowanie nowego strumienia

Zanim zostaną przesłane jakiekolwiek dane aplikacyjne, musi zostać utworzony nowy strumień i muszą być przesłane odpowiednie metadane, takie jak priorytet strumienia i nagłówki HTTP. W protokole HTTP 2.0 nowe strumienie mogą być zainicjowane zarówno przez klienta, jak i serwer, a więc należy rozważyć dwa przypadki:

- Klient inicjuje nowe żądanie, wysyłając ramkę `HEADERS` (patrz rysunek 12.7), zawierającą jednolity nagłówek z identyfikatorem strumienia, opcjonalnym 31-bitowym priorytetem, a w polu danych zestaw nagłówków HTTP z parami klucz-wartość.
- Serwer inicjuje wypychany strumień, wysyłając ramkę `PUSH_PROMISE`, która jest praktycznie identyczna z ramką `HEADERS`, z tym jednym wyjątkiem, że zawiera dodatkowy „identyfikator obiecanego strumienia", a nie jego priorytet.



Rysunek 12.7. Ramka HEADERS z opcjonalnym priorytetem

Ramki obu typów są używane do przesyłania jedynie metadanych opisujących każdy nowy strumień, a właściwe dane są dostarczane osobno w ramkach typu `DATA`. Ponadto, ponieważ obie strony mogą inicjować nowe strumienie, ich numeracja jest rozdzielona: strumienie inicjowane przez klienta mają parzyste identyfikatory, a inicjowane przez serwer — nieparzyste. Taki podział eliminuje kolizje w numeracji strumieni. Każda strona ma własny licznik zwiększany podczas inicjowania nowego strumienia.

Ponieważ metadane i dane aplikacyjne są dostarczane osobno, oznacza to, że zarówno klient, jak i serwer mogą nadawać im różne priorytety, na przykład „ruch sterujący" może być przesyłany z wyższym priorytetem, a sterowanie przepływem może dotyczyć tylko ramek typu `DATA`.

Przesyłanie danych aplikacyjnych

Po utworzeniu nowego strumienia i przesłaniu nagłówków HTTP do przesłania danych aplikacyjnych (jeżeli takie istnieją) są stosowane ramki `DATA` (patrz rysunek 12.8). Dane są dzielone na wiele ramek, z których ostatnia ma w nagłówku ustawioną flagę `END_STREAM`, oznaczającą koniec komunikatu.

Bit	+0...7	+8...15	+16...23	+24...31
0	Długość		Typ (0)	Flagi
32	R	Identyfikator strumienia		
...	*Dane HTTP*			

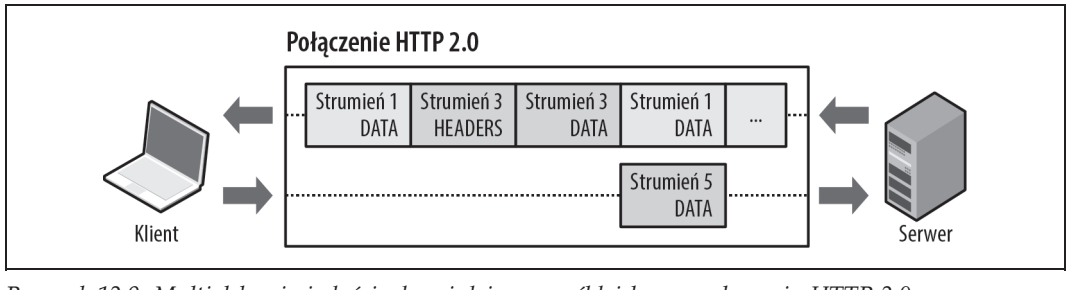
Rysunek 12.8. Ramka typu DATA

Dane nie są dodatkowo kodowane ani kompresowane. Na aplikację lub serwer spada wybór mechanizmu kodowania, np. użycia formatu zwykłego tekstu, kompresji gzip, kompresji obrazów lub wideo. Nic więcej nie można powiedzieć na temat ramki `DATA`! Cała ramka składa się z jednolitego 8-bajtowego nagłówka i następujących po nim danych HTTP.

Z technicznego punktu widzenia pole `Długość` w ramce `DATA` umożliwia przesłanie do 2^{16}–1 (65 535) bajtów danych. Jednak aby zmniejszyć blokowanie początku kolejki, standard HTTP 2.0 wymaga, aby długość ramki nie przekroczyła 2^{14}–1 (16 383). Komunikaty, które przekraczają ten limit, muszą być podzielone na kilka ramek `DATA`.

Analiza przepływu ramek DATA w protokole HTTP 2.0

Wyposażeni w podstawowe informacje na temat różnych typów ramek możemy ponownie przyjrzeć się diagramowi (patrz rysunek 12.9), który widzieliśmy wcześniej w podrozdziale „Multipleksacja żądań i odpowiedzi”, i przeanalizować przepływ danych.



Rysunek 12.9. Multipleksacja żądań i odpowiedzi we współdzielonym połączeniu HTTP 2.0

- Są trzy aktywne strumienie o numerach 1, 3 i 5.
- Wszystkie strumienie mają nieparzyste identyfikatory, ponieważ zostały zainicjowane przez klienta.
- W tej wymianie danych nie są stosowane strumienie zainicjowane przez serwer.
- Serwer wysyła kilka ramek `DATA` w strumieniu nr 1 służącym do przesyłania odpowiedzi aplikacji na żądanie klienta. Oznacza to również, że wcześniej została przesłana ramka `HEADERS`.
- Serwer wymieszał ramki `HEADERS` i `DATA` ze strumienia 3 z ramkami `DATA` ze strumienia 1 — multipleksacja odpowiedzi w akcji!
- Klient przesyła ramkę `DATA` w strumieniu 5, co oznacza, że wcześniej została przesłana ramka `HEADERS`.

Krótko mówiąc, przedstawione wyżej połączenie multipleksuje trzy równoległe strumienie na różnych etapach przetwarzania. Kolejność ramek określa serwer; nie musimy się troszczyć o typ i zawartość ramek w każdym strumieniu. Strumień 1 może przesyłać wiele danych lub sygnał wideo, ale nie będzie blokował innych strumieni wewnątrz współdzielonego połączenia!

Skorowidz

D

E

F

G

H

I

J

K

L

Ł

M

N

O

P

R

S

Ś

T

U

W

X

Z

Ź

Programowanie

O'REILLY®

Wydajne aplikacje internetowe
Przewodnik

Aplikacje internetowe systematycznie wypierają swoje klasyczne odpowiedniki. Edytory tekstu, programy graficzne czy systemy CRM w wersji online nikogo już nie zaskakują. Coraz bardziej skomplikowane narzędzia dostępne za pośrednictwem przeglądarki internetowej wymagają od deweloperów znakomitej znajomości protokołów HTTP, XHR, WebSocket i nie tylko. Dzięki tej wiedzy są oni w stanie tworzyć wydajne aplikacje, które spełnią oczekiwania użytkowników.

Ta książka to najlepsze źródło informacji poświęcone protokołom internetowym. Przygotowana przez inżyniera Google'a, odpowiedzialnego za wydajność, zawiera szereg cennych informacji, które pozwolą Ci ulepszyć Twoje własne aplikacje. W trakcie lektury dowiesz się, jak osiągnąć optymalną wydajność protokołów TCP, UDP i TLS oraz jak wykorzystać możliwości sieci mobilnych 3G/4G. W kolejnych rozdziałach zaznajomisz się z historią protokołu HTTP, poznasz jego mankamenty oraz sposoby rozwiązywania problemów. Zorientujesz się też w nowościach, jakie ma wprowadzić HTTP w wersji 2.0. W końcu odkryjesz, co mogą Ci zaoferować WebSocket oraz WebRTC, a dodatkowo poznasz skuteczne techniki strumieniowania danych w internecie. Książka ta jest obowiązkową lekturą dla każdego programisty tworzącego aplikacje internetowe!

Dzięki tej książce:

- poznasz najlepsze techniki optymalizacji ruchu w sieci
- wykorzystasz potencjał sieci bezprzewodowych oraz mobilnych
- zaznajomisz się z historią protokołu HTTP i jego mankamentami
- dowiesz się, jak nawiązać połączenie peer-to-peer za pomocą WebRTC
- zbudujesz wydajną aplikację internetową

Poznaj niuanse pozwalające na zbudowanie szybkiej aplikacji internetowej!

ISBN 978-83-246-8894-4

Cena 59,00 zł

9 788324 688944