

Systemy rozproszone | Technologie middleware, cz. II

Łukasz Czekierda, Instytut Informatyki AGH (luke@agh.edu.pl)

1. Przygotowanie do zajęć i weryfikacja środowiska

Co będzie potrzebne:

- Java
- IDE: Eclipse, IntelliJ
- wireshark z możliwością przechwytywania pakietów przechodzących przez interfejs loopback (jak na poprzednich zajęciach)
- **kompilator** Protocol Buffers (protoc)
- **wtyczka** gRPC do kompilatora protoc

Weryfikacja czy wszystko jest gotowe na zajęcia:

- Poprawne wykonanie komendy (wersja dla Windows): **protoc.exe -I=. --java_out=gen --plugin=protoc-gen-grpc-java=protoc-gen-grpc-java-1.37.0-windows-x86_64.exe --grpc-java_out=gen test.proto** (może być konieczne wskazanie ścieżki plików .exe)

2. Wykonanie ćwiczenia

2.1 Wprowadzenie

Prowadzący wprowadza Studentów w temat ćwiczenia sprawdzając równocześnie ich przygotowanie.

2.2 Podstawy protokołu HTTP/2

- 1) Włącz **wireshark** (w trybie *non-promiscuous*). Dodaj filtr **ip.addr==149.156.97.0/24**.
- 2) Używając przeglądarki **Chrome** otwórz stronę **http://www.informatyka.agh.edu.pl**. Dlaczego finalnie został użyty protokół HTTPS? Która ze stron zażądała zmiany i w jaki sposób?

- 3) Przeanalizuj ustanawianie komunikacji TLS w wireshark: szukaj pola **Client Hello** a w nim **Application Layer Protocol Negotiation**. Co tu jest negocjowane?

- 4) Aktywuj podgląd komunikacji w przeglądarce (**Ctrl-Shift-J**) i włącz prezentację wartości pól zaznaczonych na poniższym rzucie ekranu (lista aktywna po kliknięciu na wiersz Name-Method...)

```

❏ Extension: SessionTicket TLS
❏ Extension: Application Layer Protocol Negotiation
    Type: Application Layer Protocol Negotiation (0x0010)
    Length: 14
    ALPN Extension Length: 12
    ❏ ALPN Protocol
        ALPN string length: 2
        ALPN Next Protocol: h2
        ALPN string length: 8
        ALPN Next Protocol: http/1.1
❏ Extension: status_request

```

The screenshot shows the Chrome DevTools Network tab. At the top, there are tabs for Elements, Console, Sources, Network (selected), Memory, Application, Security, and Audits. Below these are checkboxes for 'Preserve log', 'Disable cache', and 'Online'. A search filter is set to 'All'. The top of the network panel shows a timeline from 0 to 11000 ms. The main area displays a list of network requests with columns: Name, Method, Status, Protocol, Scheme, Remote Address, Type, Initiator, and Time. The requests are as follows:

Name	Method	Status	Protocol	Scheme	Remote Address	Type	Initiator	Time
eng.png?76de...	GET	200	h2	https	149.156.10...	png	(index)	0 ms
footer_build...	GET	200	h2	https	149.156.10...	jpeg	(index)	0 ms
facebook_bg...	GET	200	h2	https	149.156.10...	jpeg	(index)	0 ms
www-embed...	GET	200	h2	https	172.217.16...	script	?listType=...	143 ms
base.js	GET	200	h2	https	172.217.16...	script	?listType=...	212 ms
dataimage/...	GET	200	data	data		gif	jquery.min.js	0 ms
prev.png	GET	200	h2	https	104.17.65.4...	png	jquery.min.js	0 ms
next.png	GET	200	h2	https	104.17.65.4...	png	jquery.min.js	0 ms
loading.gif	GET	200	h2	https	104.17.65.4...	gif	jquery.min.js	0 ms
close.png	GET	200	h2	https	104.17.65.4...	png	jquery.min.js	0 ms
arrow.svg?76de...	GET	200	h2	https	149.156.10...	svg+xml	jquery.min.js	1 ms

On the right side of the network panel, there is a 'Waterfall' view showing the timing of each request. The 'Time' column in the table shows the duration of each request, with some requests taking 143 ms or 212 ms. The 'Waterfall' view shows the sequence of requests and their relative timing.

- 5) Ponownie załaduj stronę WWW i sprawdź, która wersja protokołu HTTP jest wykorzystywana.
- 6) Co oznacza **h3-29** (lub podobne) w kolumnie **Protocol** dla niektórych wywołań?
- 7) Znajdź dwie strony WWW, dla których obsługa działa protokół HTTP/2 i dwie, dla których jest nadal używane HTTP/1.1. Czy główna strona AGH obsługuje HTTP/2? Użyj np. **<https://http2.pro>** lub sprawdź sam(a).

2.3 Serializacja Protocol Buffers

1. Otwórz plik **person.proto** i zapoznaj się z jego zawartością.
2. Skompiluj plik z definicją interfejsu: otwórz okno konsolowe i z poziomu głównego katalogu projektu wydaj polecenie (wersja dla Windows) **protoc.exe -I=. --java_out=gen person.proto**. Ze względu na poprawność kompilacji całości projektu, wykonaj także kompilację opisaną w punktach 2.4.3 i 2.4.11.
3. Zapoznaj się z wygenerowanymi plikami.
4. Skompiluj plik ponownie żądając generacji kodu dla wybranych innych języków programowania (**--ruby_out, --python_out, --cpp_out, ...**).
5. Sprawdź zgrubnie czas serializacji pojedynczej przykładowej wiadomości tego typu wykonując w pętli odpowiednio dużą liczbę serializacji – tak, by dało się wyznaczyć czas trwania pojedynczej.
6. Porównaj czas i efektywność (wielkość) serializacji Protocol Buffers z domyślną serializacją Java – kod w pliku **JavaSerialization.java**. Która jest szybsza? Która serializuje na mniejszej liczbie bajtów?
7. Co zawiera 2 i 32 bajt pliku serializacji proto?
8. Dodaj do definicji **person.proto** nową (dowolną) wiadomość przesyłającą sekwencję liczb niecałkowitych (oznaczającą np. wysokość przychodów osoby w ostatnich miesiącach). Użyj słowa kluczowego **repeated**. Ponownie skompiluj i przeprowadź serializację nowej wersji wiadomości zawierającej np. trzy liczby w sekwencji. O ile zwiększyła się długość wiadomości?

2.4 gRPC

1. Zaimportuj projekt do IDE.
2. **Analiza interfejsu.** Zapoznaj się z definicją interfejsu zawartą w pliku **calculator.proto**. Zawiera on nie tylko definicję wiadomości, ale i ...
3. **Kompilacja definicji interfejsu.** Skompiluj plik z definicją interfejsu: otwórz okno konsolowe i z poziomu głównego katalogu projektu wydaj polecenie (wersja dla Windows) **protoc.exe -I=. --java_out=gen --plugin=protoc-gen-grpc-java=protoc-gen-grpc-java-1.37.0-windows-x86_64.exe --grpc-java_out=gen calculator.proto**
4. Jeśli IDE nie realizuje automatycznego odświeżania w razie zmian zawartości projektu na dysku, wymuś jego odświeżenie. Występujące wcześniej błędy kompilacji powinny zniknąć. W przypadku korzystania z **IntelliJ**, po kompilacji interfejsu oznacz folder **gen** jako **Generated Sources Root**.
5. **Analiza kodu.** Przeanalizuj wygenerowane pliki źródłowe. Zaobserwuj m.in. sposób pozyskania referencji do zdalnej usługi w aplikacji klienckiej oraz różne typy tych referencji. (dla kalkulatora – trzy).
6. **Uruchomienie aplikacji.** Uruchom klienta i serwer oraz przetestuj poprawność działania aplikacji.
7. **Analiza komunikacji sieciowej.** Prześledź komunikację pomiędzy klientem i serwerem korzystając z wireshark. Jaki protokół komunikacji jest wykorzystywany? Wcześniej włącz w wireshark odpowiednie dekodowanie tego protokołu (**decode as...**)
8. **Wywołania asynchroniczne.** Prześledź i przetestuj obsługę długotrwałych wywołań (**async-add** i **future-add**).
9. **Rozbudowa interfejsu.** Do interfejsu **Calculator** dodaj nową operację mnożącą N liczb i zwracającą ich iloczyn. Zaimplementuj ją i przetestuj działanie aplikacji.
10. **Podejście obiektowe czy usługowe?** Zaobserwuj (testując), czy jest możliwe udostępnienie dla zdalnych wywołań kilku usług implementujących a) ten sam b) różne interfejsy IDL naraz - rozbudowując serwer by obsługiwał kolejną usługę przez dodanie w jego kodzie kolejnego **.addService**.
11. **Kompilacja definicji interfejsu.** Zapoznaj się z zawartością pliku **streaming.proto** i skompiluj go analogicznie jak poprzednio.
12. **Strumieniowanie przez serwer (server-side).** Wywołaj operację **generatePrimeNumbers** - zaobserwuj strumieniowanie. Narysuj diagram interakcji HTTP/2 pomiędzy klientem i serwerem. Czy to podejście ułatwia prowadzenie komunikacji w środowiskach gdzie klient jest „za NATem”? W jaki sposób (wireshark) jest sygnalizowane zakończenie wywołania strumieniowego?
13. **Strumieniowanie przez klienta (client-side).** Wywołaj operację **countPrimeNumbers** - zaobserwuj strumieniowanie. Narysuj diagram interakcji HTTP/2 pomiędzy klientem i serwerem. Czy to podejście ułatwia prowadzenie komunikacji w środowiskach gdzie klient jest „za NATem”?
14. **Równoległość wywołań.** Zaobserwuj (wireshark) jaki mechanizm HTTP/2 wykorzystuje gRPC do multipleksacji żądań. W tym celu zainicjuj wiele wywołań wykonujących się równocześnie (oczywiście u tego samego klienta).
15. **Ping.** Prześledź która ze stron i kiedy wysyła pakiety PING (HTTP2). Po co są one wysyłane? W razie chęci zmiany tego zachowania spójrz tu: https://grpc.github.io/grpc/cpp/md_doc_keepalive.html oraz tu: <https://github.com/grpc/grpc-java/issues/7237>. Zmiana parametrów po stronie serwera wymaga wcześniejszej wymiany **ServerBuilder** na **NettyServerBuilder**.

16. Analiza ruchu sieciowego. Pliki **grpc-1.pcapng** i **grpc-2.pcapng** zawierają zapis przykładowej komunikacji. Prześledź interesujące Cię aspekty komunikacji. Ciekawsze rzeczy to:

- identyfikatory strumieni HTTP/2
- różne typy ramek HTTP/2
- opóźnienie wywołania np. Add (różnica czasu pomiędzy żądaniem a odpowiedzią) w grpc-1.pcapng
- wywołanie z określonym i przekroczonym deadline (100 ms) (strumień #11 w grpc-1.pcapng) – gdzie ta wartość 100 ms została podana?
- wywołanie strumieniowe strony serwerowej (strumień #3 w grpc-2.pcapng). Skąd klient wie, że strumień się zakończył?
- wywołanie strumieniowe strony klienckiej (strumień #5 w grpc-2.pcapng). Skąd serwer wie, że strumień się zakończył?