

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: IntelliJ
- **wireshark** z możliwością przechwytywania pakietów przechodzących przez interfejs loopback
- **kompilator** Protocol Buffers (protoc) i **wtyczka** gRPC do kompilatora protoc
- **nginx**

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-1.54.0-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

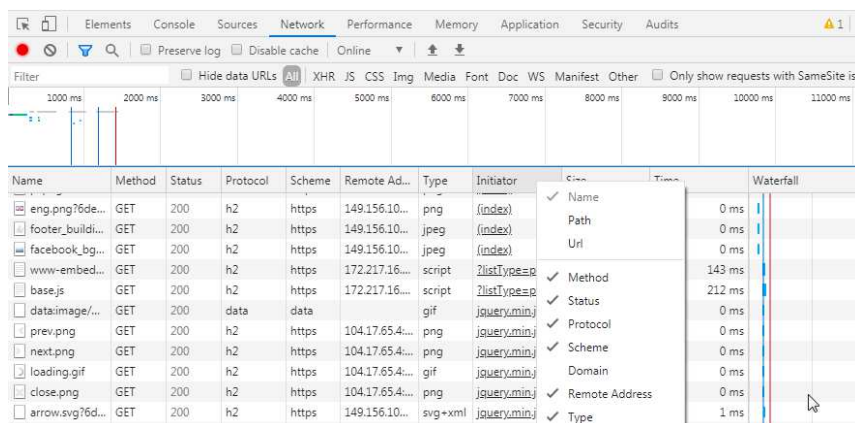
- 1) Włącz **wireshark** (w trybie *non-promiscuous*). Dodaj filtr `ip.addr==149.156.97.0/24`.
- 2) Używając przeglądarki **Chrome** otwórz stronę <http://www.informatyka.agh.edu.pl>. Dlaczego finalnie został użyty protokół HTTPS? Która ze stron zażądała zmiany i w jaki sposób? Jeśli przeglądarka jest „mądrzejsza” i od razu próbuje użyć protokołu HTTPS, możesz użyć polecenia `ncat`:

```
ncat --crlf www.informatyka.agh.edu.pl 80
GET / HTTP/1.1
Host: www.informatyka.agh.edu.pl
```

```
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
```

- 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 (**F12**) i włącz prezentację wartości pól zaznaczonych na poniższym rzucie ekranu (lista aktywna po kliknięciu na wiersz Name-Method...)



- 5) Ponownie załaduj stronę WWW i sprawdź, która wersja protokołu HTTP jest wykorzystywana.
- 6) Co oznacza **h3** w kolumnie **Protocol** dla niektórych wywołań?
- 7) Znajdź dwie strony WWW, dla których obsługi 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. Zaimportuj wskazany projekt do IDE.
2. Otwórz plik **person.proto** i zapoznaj się z jego zawartością.
3. 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 2.4.2 i 2.4.12.
4. Zapoznaj się z wygenerowanymi plikami (katalog **gen**).
5. Skompiluj plik ponownie żądając generacji kodu dla wybranych innych języków programowania (**--ruby_out, --python_out, --cpp_out, ...**).
6. Sprawdź zgrubnie czas serializacji pojedynczej przykładowej wiadomości tego typu wykonując w pętli odpowiednio dużą liczbę serializacji – tak, by dało się wyznaczyć czas trwania pojedynczej.
7. 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?
8. Użyj aplikacji <https://protobuf-decoder.netlify.app/> by zdekodować zakodowaną wiadomość proto. Czego ona **nie** zawiera? (Podobne informacje można uzyskać tak: **protoc --decode_raw <plik.ser**)
9. Zdekoduj zakodowaną wiadomość używając polecenia: **protoc --decode tutorial.Person person.proto <plik.ser**. Porównaj rezultaty. Z czego wynikają różnice?
10. Dodaj do definicji **person.proto** nową (dowolną) wiadomość zawierającą 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. **Analiza interfejsu.** Zapoznaj się z definicją interfejsu zawartą w pliku **calculator.proto**. Zawiera on nie tylko definicję wiadomości, ale i ...
2. **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-1.54.0-windows-x86_64.exe --grpc-java_out=gen calculator.proto**
3. Jeśli IDE nie realizuje automatycznego odświeżania w razie zmian zawartości projektu na dysku, wymuś jego odświeżenie. Występujące wcześniej błędy kompilacji powinny zniknąć.
4. **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).
5. **Uruchomienie aplikacji.** Uruchom klienta i serwer oraz przetestuj poprawność działania aplikacji.
6. **Analiza komunikacji sieciowej.** Prześledź komunikację pomiędzy klientem i serwerem korzystając z Wireshark. Jaki protokół komunikacji jest wykorzystywany? Przed analizą włącz w Wireshark odpowiednie dekodowanie pakietów tego protokołu (**decode as...**) Nie chcąc analizować potwierdzeń TCP można do filtra dodać warunek **http2**.
7. **Analiza komunikacji sieciowej.** Używając Wireshark sprawdź jaki typ danych jest użyty do zapisania liczb stanowiących argumenty i wynik wywołania procedur **add1** i **add2**. Ile bajtów zajmują te wartości?
8. **Wywołania nieblokujące.** Prześledź i przetestuj nieblokującą obsługę długotrwałych wywołań (**nonblock-add** i **future-add**).
9. **Mechanizm deadline.** Prześledź wywołania **add-deadline1** oraz **add-deadline2**. Czy jest to mechanizm kliencki czy serwerowy (jeśli serwerowy, to w którym miejscu w wiadomości jest przesyłana zadana wartość?).
10. **Rozbudowa interfejsu.** Do interfejsu **Calculator** dodaj nową operację mnożącą **N** liczb i zwracającą ich iloczyn. Zaimplementuj ją i przetestuj działanie aplikacji. Może warto przewidzieć zgłoszenie jakiegoś błędu?
11. **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**.
12. **Kompilacja definicji interfejsu.** Zapoznaj się z zawartością pliku **streaming.proto** i skompiluj go analogicznie jak poprzednio.
13. **Strumieniowanie przez serwer (server-side).** Wywołaj operację **generatePrimeNumbers (gen-prime)** - 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?
14. **Strumieniowanie przez klienta (client-side).** Wywołaj operację **countPrimeNumbers (count-prime)**- zaobserwuj strumieniowanie. Narysuj diagram interakcji HTTP/2 pomiędzy klientem i serwerem. Czy to podejście ułatwia prowadzenie komunikacji w środowiskach gdzie klient jest „za NATem”?

15. **Równoległość wywołań.** Zaobserwuj (wireshark) jaki mechanizm protokołu HTTP/2 wykorzystuje gRPC do multipleksacji żądań. W tym celu zainicjuj wiele wywołań wykonujących się (niemal) równocześnie (oczywiście u tego samego klienta). Najlepiej będzie użyć długotrwałych wywołań nieblokujących.
16. **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**.
17. **Reverse proxy.** Uruchom **nginx** (maszyna wirtualna SR: katalog **c:\Program Files\util**) z opcją **-c** wskazując plik **grpc1.conf** zawarty w podkatalogu **conf**. Uruchom dwie (równoczesne) instancje serwera gRPC zgodnie z konfiguracją (numery portów) zawartą w tym pliku. (Aby w IntelliJ uruchomić więcej niż jedną instancję procesu naraz: Run | Edit configurations | *aplikacja* | More Options | Allow multiple instances.) Zmodyfikuj konfigurację klienta gRPC by komunikował się z serwerami za pośrednictwem *reverse proxy*. Przetestuj obserwując równoważenie obciążenia. Czy wywołania strumieniowe są poprawnie obsługiwane?
18. **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?