Systemy rozproszone | Technologie middleware, cz. II

Łukasz Czekierda, Instytut Informatyki AGH (luke@agh.edu.pl) (2025)

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-1.71.0-windows-x86_64.exe --grpc-java_out=gen PLIK.proto** *(może być konieczne wskazanie ścieżki plików .exe)*

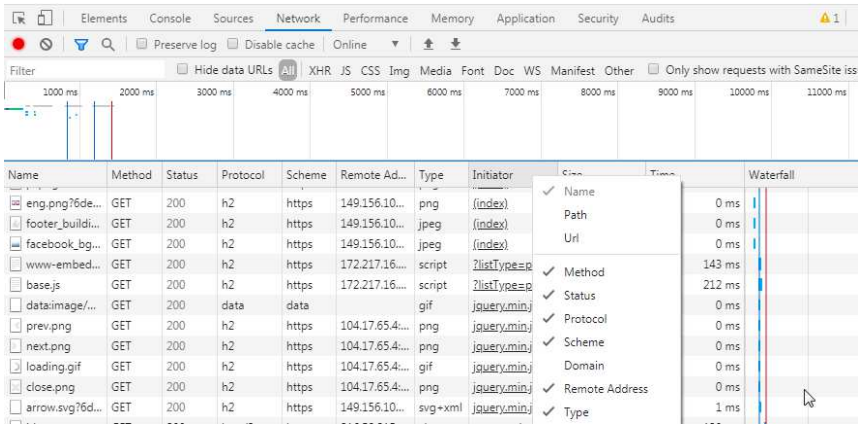
2. Wykonanie ćwiczenia

2.1 Wprowadzenie

Prowadzący wprowadza Studentów w temat ćwiczenia sprawdzając równocześnie ich przygotowanie.

2.2 Podstawy protokołu HTTP/2 (i wzmianka o HTTP/3)

1) Otwórz w Wireshark plik **informatyka.pcapng** (→UPEL) zawierający zapis komunikacji HTTP z serwerem **www.informatyka.agh.edu.pl**
2) W Wireshark otwórz okno ustawień protokołu TLS (**Edycja|Preferencje|Protokoły|TLS**) i do pola **Pre-Master-Secret log filename** wstaw lokalizację pliku **ssl.log** (→UPEL) zawierającego klucze szyfrowanej komunikacji.
3) W filtrze wyświetlania wpisz **http||http2||tls**
4) W pakiecie #8 zaobserwuj przekierowanie (**302**). Pakiety #13, #15, #17 i #19 służą do ustawienia sesji TLS. Zaobserwuj jakiego protokołu żąda klient w **Client Hello|Handshake Protocol|Application Layer Protocol Negotiation**. Jaki protokół jest protokołem drugiego wyboru klienta? Jaki protokół wybrał serwer (Server Hello, ALPN)?
5) Prześledź komunikację HTTP2. Zwróć uwagę na skompresowane nagłówki http (**Headers**) (w zakładce **Decrypted TLS** widać takie, jakie są, w zakładce **Decompressed Header** – zdekompresowane).
6) Zwróć uwagę na identyfikatory strumieni np. **[33]**. Te nieparzyste są inicjowane przed klienta, parzyste są inicjowane prze serwer. Zobacz, jak kończy się sesja HTTP (pakiet #4194).
7) Aktywuj podgląd komunikacji w przeglądarce (**F12)** i włącz prezentację wartości pól zaznaczonych na poniższym zrzucie ekranu (lista aktywna po kliknięciu na wiersz **Name-Method**…)



8) Załaduj jakąś stronę WWW i sprawdź, która wersja protokołu HTTP jest wykorzystywana.

9) Co oznacza **h3** w kolumnie **Protocol** dla niektórych wywołań?

a) Jeśli go nie widać, postaraj się znaleźć w odpowiedzi (Response headers) jakiegoś serwera (np. google) podobną wartość: **Alt-Svc: h3=":443"; ma=2592000,h3-29=":443"; ma=2592000** Co to oznacza?
b) Obsługa protokołu HTTP/3 wymaga obsługi protokołu QUIC: w Chrome: **chrome://flags/** + „Experimental QUIC protocol".

10) 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://tools.keycdn.com/http2-test** lub sprawdź w inny sposób.

2.3 Serializacja Protocol Buffers

1. Zaimportuj projekt **Serialization** 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**
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. Przeanalizuj klasę **ProtoSerialization**, w szczególności rekord danych utworzonych na podstawie definicji w pliku **proto**. Uruchom test serializacji. Dla uzyskania mierzalnych wyników czasu serializacji test wykonuje się wielokrotnie. Ile (w przeliczeniu) trwa na Twoim komputerze pojedyncza serializacja? Na ilu bajtach zostały zapisane serializowane dane?
7. Porównaj czas i efektywność (wielkość) serializacji Protocol Buffers z innymi sposobami serializacji (kod w pliku **OtherSerializations.java**). Która jest najszybsza? Która serializuje na najmniejszej 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. Zaimportuj projekt **gRPC** 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-1.71.0-windows-x86_64.exe --grpc-java_out=gen calculator.proto** (*Ze względu na poprawność kompilacji projektu wykonaj teraz także punkt 2.4.12.*)
4. Jeśli IDE nie realizuje automatycznego odświeżania w razie zmian zawartości projektu na dysku, wymuś jego odświeżenie. Występujące wcześniej błędy kompilacji powinny zniknąć.
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 podstawowego 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? Przed analizą włącz w Wireshark odpowiednie dekodowanie pakietów tego protokołu (**decode as…)** Nie chcąc analizować potwierdzeń TCP możesz do filtra dodać warunek **http2**. Jeśli komunikacja nie będzie prezentowana jako komunikacja gRPC, w **Edytuj | Preferencje | Protokoły** i zaznacz **Embed gRPC messages…** Komunikacja HTTP2/gRPC może nie być poprawnie pokazywana jeśli Wireshark nie „złapał" całej komunikacji – od rozpoczęcia połączenia TCP.
8. **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 zajmuje każda z tych wartości?
9. **Wywołania nieblokujące.** Prześledź i przetestuj nieblokującą obsługę długotrwałych wywołań (**nonblock-add** i **future-add**).
10. **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)?

11. **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?
12. **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**. Jak to było w Ice?
13. **Kompilacja definicji interfejsu.** Zapoznaj się z zawartością pliku **streaming.proto** i skompiluj go analogicznie jak poprzednio.
14. **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?
15. **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"?
16. **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 przez tego samego klienta). Najlepiej będzie użyć długotrwałych wywołań nieblokujących.
17. **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**.
18. **Reverse proxy**. Uruchom **ningx** (maszyna wirtualna SR: katalog **c:\Program Files\util**) z opcją **-c** wskazując plik **grpc1.conf** (→UPEL). 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?
19. **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?