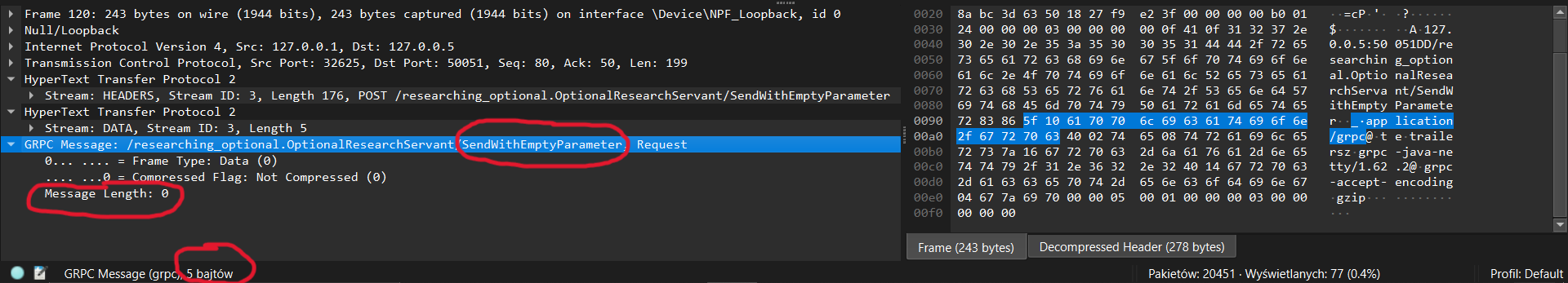
**Zadanie I.4 - Opcjonalne pola struktur danych i argumenty wywołania** middleware

# gRPC

gRPC korzysta z Protobufa jako formatu wymiany danych, dlatego analiza gRPC mocno pokrywa się z analizą sposobu działania samego Protobufa.

Zaznaczę jeszcze na wstępie, że korzystam ze składni Protobuf3.

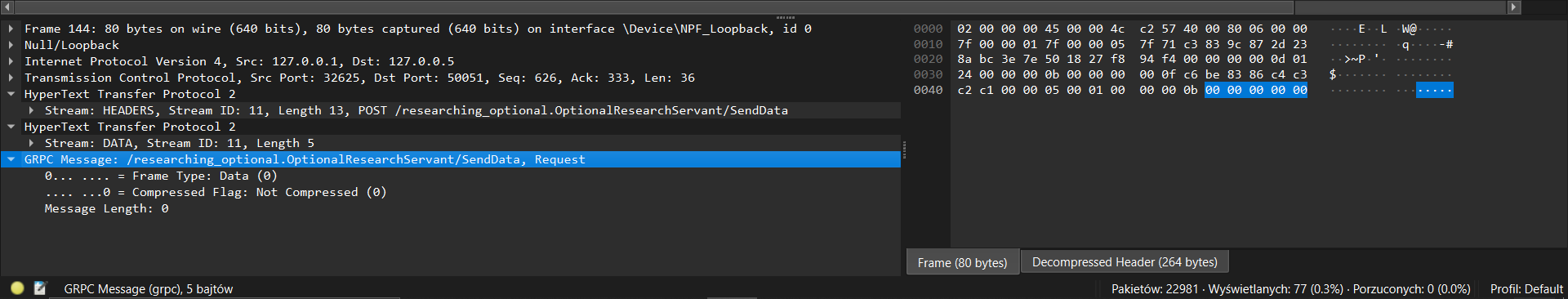
Podczas analizy gRPC najbardziej będą nas interesowały pewne części zrzutów ekranu. Zawarte są tam m.in. nazwa wywołanej usługi, zajętość wiadomości gRPC oraz zajętość struktur w tej wiadomości (HTTP2#DataStream#payload):



## Opcjonalność pól struktur danych

W protobufie3 domyślnie wszystkie pola są opcjonalne. Twórcy doszli do wniosku, że pola „required” przynoszą więcej szkody niż pożytku, więc uniemożliwili specyfikowanie pól jako obowiązkowe. Tym samym wszystkie pola automatycznie stały się opcjonalne. (jednak słówko kluczowe „optional” pozostało, ale o tym później).

### Struktura bez żadnego ustawionego pola



Taka struktura zajmuje 0B, ale gRPC message wciąż zajmuje 5B (= 1B dla Frame Type oraz Compressed Flag + 4B dla Message Length).

### Struktura w ogóle bez żadnego możliwego pola

Mowa o strukturze „message Empty {}”.

Obraz zawierający tekst, oprogramowanie, Oprogramowanie multimedialne, zrzut ekranu

Opis wygenerowany automatycznie

Sytuacja analogiczna jak w przypadku struktury z polami, ale bez żadnego ustawionego pola.

Dla porównania to samo wywołanie, ale z w pełni uzupełnioną strukturą. Zajętość jest oczywiście większa:

Obraz zawierający tekst, oprogramowanie, Oprogramowanie multimedialne, Oprogramowanie graficzne

Opis wygenerowany automatycznie

### Pola „repeated”

Pola typu „repeated” także można, w pewnie sposób, traktować jako pola opcjonalne (a w sumie jako powtórzenie danego pola od zero- do wielokrotnie).

Dla porównania (wypełnione pola):Obraz zawierający tekst, oprogramowanie, Oprogramowanie multimedialne, Oprogramowanie graficzne

Opis wygenerowany automatycznie

Powtórzenie zerokrotne:Obraz zawierający tekst, oprogramowanie, Oprogramowanie multimedialne, Oprogramowanie graficzne

Opis wygenerowany automatycznie

Jak widać, powtórzenie zerokrotne jest równoważne niedefiniowaniu żadnego pola.

### One of

Słowo kluczowe “one of” daje możliwość zdefiniowania co najwyżej jednego pola spośród wymienionych. Co istotne, użycie „one of” umożliwia zbadanie, czy dane pole zostało ustawione explicite przez klienta, czy znajduje się tam domyślna wartość.

Ustawiony int:

Obraz zawierający tekst, Oprogramowanie multimedialne, oprogramowanie, Oprogramowanie graficzne

Opis wygenerowany automatycznie

Ustawiony Empty:

Obraz zawierający tekst, oprogramowanie, Oprogramowanie multimedialne, Oprogramowanie graficzne

Opis wygenerowany automatycznie

**Ważne!** Empty zajmuje w tym wypadku miejsce!

Nic nie ustawione:

Obraz zawierający tekst, oprogramowanie, Oprogramowanie multimedialne, Oprogramowanie graficzne

Opis wygenerowany automatycznie

Wyniki metod hasSomething() dla trzech przedstawionych powyżej przykładów

Obraz zawierający tekst, zrzut ekranu, Czcionka, design

Opis wygenerowany automatycznie

Jak widać, dzięki użyciu „one of” jesteśmy w stanie rozróżnić te trzy sytuacje. Nawet sytuacje, kiedy ustawiamy strukturę Empty oraz kiedy nie ustawiamy nic.

### “Optional” jawne vs “optional” domyślne

Pomimo tego, że Protobuf3 wyeliminował słowo „required” a wszystkie pola są opcjonalne, to wciąż istnieje słowo kluczowe „optional”. Dlaczego? Ponieważ jest różnica między domyślną opcjonalnością a jawna opcjonalnością. Chociaż w sumie okazuje się, że to zależy od konkretnego języka, bo w wielu jest inaczej.

Na ten przykład, w Javie wywołanie DataExplicitlyOptional.newBuilder().setData(null) skutkuje wyjątkiem NullPointerException. Poprawnym wywołaniem w tym wypadku jest DataExplicitlyOptional.newBuilder().setData(Data.getDefaultInstance()). W niektórych językach można, a czasem nie można, stosować odpowiedniki nulla (None, missing, undefined itp.). Więcej informacji na ten temat: <https://gist.github.com/kshchepanovskyi/c0bf9b9f95a02742a310>

Spróbowałem wywołań dla pól niejawnie opcjonalnych oraz dla jawnie opcjonalnych (ze słowem „optional”).

Pola domyślnie opcjonalne ustawione na wartości puste/domyślne:Obraz zawierający oprogramowanie, Oprogramowanie multimedialne, tekst, Oprogramowanie graficzne

Opis wygenerowany automatycznie

Pola domyślnie opcjonalne nie ustawiane na żadne wartości:Obraz zawierający tekst, oprogramowanie, Oprogramowanie multimedialne, Ikona komputerowa

Opis wygenerowany automatycznie

Pola domyślnie opcjonalne. Działanie metod „hasField()”:

Obraz zawierający tekst, zrzut ekranu, Czcionka

Opis wygenerowany automatycznie

Pola jawnie opcjonalne ustawione na wartości puste/domyślne:Obraz zawierający tekst, oprogramowanie, Oprogramowanie multimedialne, Oprogramowanie graficzne

Opis wygenerowany automatycznie

Pola jawnie opcjonalne nie ustawiane na żadne wartości :Obraz zawierający tekst, oprogramowanie, Oprogramowanie multimedialne, Ikona komputerowa

Opis wygenerowany automatycznie

Pola domyślnie opcjonalne. Działanie metod „hasField()”:Obraz zawierający tekst, zrzut ekranu, Czcionka

Opis wygenerowany automatycznie

Jak widać, różnica jest znacząca.

Po pierwsze, dla pól domyślnie opcjonalnych nie istnieją wygenerowane metody sprawdzające obecność pól dla typów prymitywnych. Dla pól jawnie domyślnych jest to możliwe. Wynika to z tego, że pod spodem słowo kluczowe „optional” jest zamieniane na „oneof”, a, jak przedstawiłem we wcześniejszym podpunkcie, użycie „oneof” zapewnia metody typu „hasField” także dla typów prymitywnych

Po drugie, warto pamiętać, że jeśli chcemy „usunąć” jakieś pole („odustawić” jego wartość?) i zrobimy to ustawiając wartość domyślną (np. 0 dla inta, pusty ciąg znaków dla stringa, niewypełnioną strukturę dla struktury), to:

* dla opcjonalności niejawnej wartości prymitywne „znikną z przewodu”, ale struktury już nie
* dla opcjonalności jawnej zarówno wartości prymitywne jak i struktury pozostaną „w przewodzie”.

Z tego też powodu do tego celu zaleca się używanie metod „clearField”.

Więcej na ten temat tutaj: <https://github.com/protocolbuffers/protobuf/blob/v3.12.0/docs/field_presence.md>

## Opcjonalność argumentów wywołań

Definiowanie serwisów w plikach .proto nie pozwala na używanie czegoś, co miałoby oznaczać brak wartości podawanej (parametru) lub zwracanej (return), tzn. nie istnieje słówko „void”. Każde odwołanie do usługi wymaga podania argumentów oraz otrzymania odpowiedzi, nawet jeśli nas ona nie interesuje (chociaż możemy zignorować odpowiedź wysyłając zapytanie nieblokujące, ale dane i tak zostaną przesłane klientowi).

Oczywiście, w sytuacji, gdy nie musimy podać argumentów lub nie potrzebujemy wartości zwracanej, możemy skorzystać ze specjalnie przygotowanej przez Google’a struktury Empty w pliku "google/protobuf/empty.proto". Lub też, jak ja, możemy ją sami napisać, bo jest to po prostu „message Empty {}”.

Wywołanie usługi z argumentem „Empty”:

Obraz zawierający tekst, oprogramowanie, Oprogramowanie multimedialne, zrzut ekranu

Opis wygenerowany automatycznie

Odpowiedź na powyższe zapytanie:

Obraz zawierający tekst, oprogramowanie, Oprogramowanie multimedialne, zrzut ekranu

Opis wygenerowany automatycznie

Jak widać, sytuacja jest analogiczna jak podczas badania samej struktury „Empty”, tzn. sama struktura zajmuje 0B, ale żeby gRPC mogło ją przesłać, trzeba wykorzystać 5B.

# Ice

## Opcjonalność pól struktur danych

W przypadku Slice’a struktury nie mogą mieć opcjonalnych pól. Jedynymi typami, które mogą mieć opcjonalne pola są:

* klasy,
* wyjątki.

### „Optional” w klasach

Z uzupełnionym opcjonalnym polem:Obraz zawierający tekst, oprogramowanie, Oprogramowanie multimedialne, Oprogramowanie graficzne

Opis wygenerowany automatycznie

Bez opcjonalnego pola:Obraz zawierający tekst, oprogramowanie, Oprogramowanie multimedialne, Oprogramowanie graficzne

Opis wygenerowany automatycznie

Faktycznie, brak ustawionej wartości w opcjonalnym polu powoduje przesył mniejszej ilości bajtów. Zgadza się to z opisem w dokumentacji: <https://doc.zeroc.com/ice/3.7/best-practices/optional-values>

### „Optional” w wyjątkach

Z ustawioną wartością:Obraz zawierający tekst, oprogramowanie, Oprogramowanie multimedialne, Oprogramowanie graficzne

Opis wygenerowany automatycznie

Bez ustawionej wartości: Obraz zawierający tekst, oprogramowanie, Oprogramowanie multimedialne, Oprogramowanie graficzne

Opis wygenerowany automatycznie

Co ciekawe, oba wyjątki zajmują taką samą liczbę bajtów.

### Puste sekwencje

Ustawione dwie wartości: Obraz zawierający zrzut ekranu, tekst, Oprogramowanie multimedialne

Opis wygenerowany automatycznie

Ustawiona jedna wartość: Obraz zawierający tekst, Oprogramowanie multimedialne, oprogramowanie, Oprogramowanie graficzne

Opis wygenerowany automatycznie

Brak ustawionych wartości: Obraz zawierający tekst, Oprogramowanie multimedialne, oprogramowanie, Oprogramowanie graficzne

Opis wygenerowany automatycznie

Zgodnie z oczekiwaniami, im więcej liczb w sekwencji, tym większej liczby bajtów potrzenujemy (za każdym razem o 4 więcej dla intów).

## Opcjonalność argumentów wywołań

### „void” jako parametr metody lub wartość zwracana

Obraz zawierający tekst, oprogramowanie, Oprogramowanie multimedialne, Oprogramowanie graficzne

Opis wygenerowany automatycznie

### „Optional” w parametrach metody lub w zwracanej wartości

Ustawione parametry:Obraz zawierający tekst, Oprogramowanie multimedialne, oprogramowanie, Oprogramowanie graficzne

Opis wygenerowany automatycznie

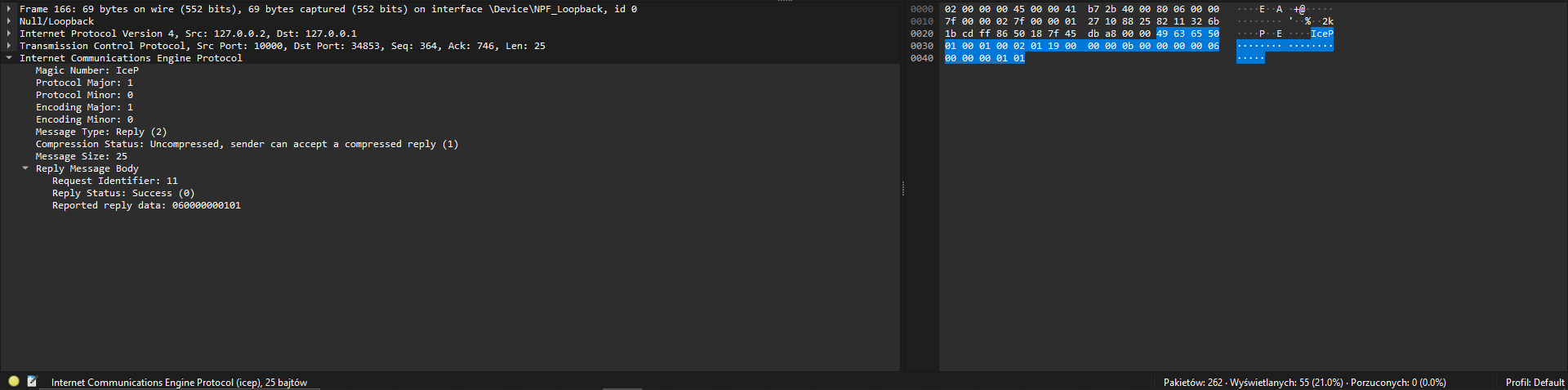
Nieustawione parametry: Obraz zawierający tekst, Oprogramowanie multimedialne, oprogramowanie, Oprogramowanie graficzne

Opis wygenerowany automatycznie

Zajętość wiadomości zmniejszyła się.

Ustawione wartości zwracane:Obraz zawierający tekst, Oprogramowanie multimedialne, oprogramowanie, Oprogramowanie graficzne

Opis wygenerowany automatycznie

Nieustawione wartości zwracane:

Tutaj, analogicznie, zajętość wiadomości zmniejszyła się