Mateusz Skowron

Opcjonalne wartości struktur danych i argumenty wywołania middleware

1. **ICE**

   W przypadku niektórych technologii middleware, takich jak ICE, domyślnie wszystkie pola w strukturach i jako argumenty są wymagane. Aby oznaczyć pole jako opcjonalne, należy zdefiniować je jako "optional" wraz z tagiem.Jeśli wartość opcjonalna nie jest ustawiona (ma wartość domyślną lub jest niezainicjowana), to nie jest przekazywana wraz z przestrzenią nazw (np. w nagłówku komunikatu). W tym przypadku nie ma potrzeby przekazywania tagu do przestrzeni nazw, ponieważ wartość jest znana i stała.
   Jeśli jednak wartość opcjonalna jest ustawiona, to jej tag jest przekazywany wraz z wartością do przestrzeni nazw. Dzięki temu odbiorca wiadomości wie, że wartość jest opcjonalna i może ją obsłużyć w odpowiedni sposób.
   Dzięki takiemu mechanizmowi, nieustawione pola opcjonalne są pominięte podczas serializacji i nie są wysyłane przez sieć. Natomiast pola, których wartości opcjonalne są ustawione dodają dodatkowy narzut, aby oznaczyć pole jako ustawione.

   W Wireshark można zaobserwować różnice w rozmiarze pakietów sieciowych, gdy pole opcjonalne jest ustawione i gdy nie jest. W przypadku braku ustawienia wartości opcjonalnej, pole to nie jest wysyłane przez sieć, co prowadzi do mniejszego rozmiaru pakietu.

   Przykładowy IDL, który posłuży do testów:

```
module Bookstore
{
    class BookOptional
    {
        string title;
        string author;
        optional(1) int year;
    };

    class BookNoOptional
    {
        string title;
        string author;
        int year;
    };

    interface BookstoreService
    {
        bool addBookOptional(string title, string author, optional(1) int year);
        bool addBookNoOptional(string title, string author, int year);
        bool addBookStructOptional(BookOptional book);
        bool addBookStructNoOptional(BookNoOptional book);
    };
};
```

Uruchamiamy kod testujący:

```
// Just arguments
boolean res1 = service.addBookNoOptional( title: "Harry Potter",  author: "J.K Rowling",  year: 1997);
System.out.println(res1);

boolean res2 = service.addBookOptional( title: "Harry Potter",  author: "J.K Rowling", java.util.OptionalInt.empty());
System.out.println(res2);

boolean res3 = service.addBookOptional( title: "Harry Potter",  author: "J.K Rowling",  year: 1997);
System.out.println(res3);

// Structs
boolean res4 = service.addBookStructNoOptional(new BookNoOptional ( title: "Harry Potter",  author: "J.K Rowling",  year: 1997));
System.out.println(res4);

boolean res5 = service.addBookStructOptional(new BookOptional( title: "Harry Potter",  author: "J.K Rowling"));
System.out.println(res5);

boolean res6 = service.addBookStructOptional(new BookOptional( title: "Harry Potter",  author: "J.K Rowling",  year: 1997));
System.out.println(res6);
```

W Wiresharku analizujemy pakiety:

ip.addr == 127.0.0.2

No.	Time	Source	Destination	Protocol	Length	Info
688	31.776091	127.0.0.1	127.0.0.2	TCP	44	1992 → 10000 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
741	33.820730	127.0.0.1	127.0.0.2	TCP	56	2020 → 10000 [SYN] Seq=0 Win=65535 Len=0 MSS=65495 WS=4 SAC
742	33.820812	127.0.0.2	127.0.0.1	TCP	56	10000 → 2020 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=654
743	33.820846	127.0.0.1	127.0.0.2	TCP	44	2020 → 10000 [ACK] Seq=1 Ack=1 Win=131072 Len=0
744	33.821309	127.0.0.2	127.0.0.1	ICEP	58	Validate connection
745	33.821349	127.0.0.1	127.0.0.2	TCP	44	2020 → 10000 [ACK] Seq=1 Ack=15 Win=131056 Len=0
750	33.861341	127.0.0.1	127.0.0.2	ICEP	120	Request(1): bookstore.ice_isA()
751	33.861402	127.0.0.2	127.0.0.1	TCP	44	10000 → 2020 [ACK] Seq=15 Ack=77 Win=130996 Len=0
752	33.861609	127.0.0.2	127.0.0.1	ICEP	70	Reply(1): Success
753	33.861631	127.0.0.1	127.0.0.2	TCP	44	2020 → 10000 [ACK] Seq=77 Ack=41 Win=131032 Len=0
754	33.867160	127.0.0.1	127.0.0.2	ICEP	129	Request(2): bookstore.addBookNoOptional()
755	33.867226	127.0.0.2	127.0.0.1	TCP	44	10000 → 2020 [ACK] Seq=41 Ack=162 Win=130908 Len=0
756	33.868010	127.0.0.2	127.0.0.1	ICEP	70	Reply(2): Success
757	33.868079	127.0.0.1	127.0.0.2	TCP	44	2020 → 10000 [ACK] Seq=162 Ack=67 Win=131004 Len=0
758	33.871007	127.0.0.1	127.0.0.2	ICEP	123	Request(3): bookstore.addBookOptional()
759	33.871056	127.0.0.2	127.0.0.1	TCP	44	10000 → 2020 [ACK] Seq=67 Ack=241 Win=130832 Len=0
760	33.871633	127.0.0.2	127.0.0.1	ICEP	70	Reply(3): Success
761	33.871685	127.0.0.1	127.0.0.2	TCP	44	2020 → 10000 [ACK] Seq=241 Ack=93 Win=130980 Len=0
762	33.874333	127.0.0.1	127.0.0.2	ICEP	128	Request(4): bookstore.addBookOptional()
763	33.874386	127.0.0.2	127.0.0.1	TCP	44	10000 → 2020 [ACK] Seq=93 Ack=325 Win=130748 Len=0
764	33.874873	127.0.0.2	127.0.0.1	ICEP	70	Reply(4): Success
765	33.874905	127.0.0.1	127.0.0.2	TCP	44	2020 → 10000 [ACK] Seq=325 Ack=119 Win=130952 Len=0
766	33.885367	127.0.0.1	127.0.0.2	ICEP	165	Request(5): bookstore.addBookStructNoOptional()
767	33.885448	127.0.0.2	127.0.0.1	TCP	44	10000 → 2020 [ACK] Seq=119 Ack=446 Win=130624 Len=0
768	33.886286	127.0.0.2	127.0.0.1	ICEP	70	Reply(5): Success
769	33.886326	127.0.0.1	127.0.0.2	TCP	44	2020 → 10000 [ACK] Seq=446 Ack=145 Win=130928 Len=0
770	33.889546	127.0.0.1	127.0.0.2	ICEP	157	Request(6): bookstore.addBookStructOptional()
771	33.889584	127.0.0.2	127.0.0.1	TCP	44	10000 → 2020 [ACK] Seq=145 Ack=559 Win=130512 Len=0
772	33.890362	127.0.0.2	127.0.0.1	ICEP	70	Reply(6): Success
773	33.890391	127.0.0.1	127.0.0.2	TCP	44	2020 → 10000 [ACK] Seq=559 Ack=171 Win=130900 Len=0
774	33.892788	127.0.0.1	127.0.0.2	ICEP	163	Request(7): bookstore.addBookStructOptional()
775	33.892837	127.0.0.2	127.0.0.1	TCP	44	10000 → 2020 [ACK] Seq=171 Ack=678 Win=130392 Len=0
776	33.893828	127.0.0.2	127.0.0.1	ICEP	70	Reply(7): Success
777	33.893867	127.0.0.1	127.0.0.2	TCP	44	2020 → 10000 [ACK] Seq=678 Ack=197 Win=130876 Len=0

1. Argumenty
    - Podanie wszystkich argumentów do metody *addBookNoOptional*
      TCP Payload wynosi 85 bajtów.
      Input parameters size wynosi 35 bajtów.
    - Nie podanie opcjonalnego argumentu do metody *addBookOptional*
      TCP Payload wynosi 79 bajtów.
      Input parameters size wynosi 31 bajtów.
    - Podanie wszystkich argumentów do metody *addBookOptional*
      TCP Payload wynosi 84 bajtów.
      Input parameters size wynosi 36 bajtów.

2. Struktury
    - Podanie wszystkich argumentów do metody *addBookStructNoOptional*
      TCP Payload wynosi 121 bajtów.
      Input parameters size wynosi 65 bajtów.
    - Nie podanie opcjonalnego argumentu do metody *addBookStructOptional*
      TCP Payload wynosi 113 bajtów.
      Input parameters size wynosi 59 bajtów.
    - Podanie wszystkich argumentów do metody *addBookStructOptional*
      TCP Payload wynosi 119 bajtów.
      Input parameters size wynosi 65 bajtów.

Z powyższych wyników wynika, że używanie opcjonalnych pól w technologiach middleware, takich jak ICE, może pomóc zmniejszyć rozmiar pakietów sieciowych, co prowadzi do efektywniejszego przesyłania danych. W przypadku gdy wartość opcjonalna nie jest ustawiona, pole to jest pomijane podczas serializacji i nie jest wysyłane przez sieć, co prowadzi do mniejszego rozmiaru pakietu. Natomiast w przypadku gdy wartość opcjonalna jest ustawiona, jej tag jest przekazywany wraz z wartością do przestrzeni nazw, co pozwala na obsłużenie jej przez odbiorcę w odpowiedni sposób. W ten sposób narzut na oznaczenie pola jako ustawione jest minimalny.

2. Thrift

W przypadku Apache Thrift, opcjonalne wartości można używać tylko w strukturach danych IDL. Można je zdefiniować przy pomocy słowa kluczowego "optional". W przypadku argumentów nie da się utworzyć pól opcjonalnych. Niezależnie od tego, czy opcjonalne parametry są ustawione, czy nie, rozmiar przesyłanych danych pozostanie taki sam. Nie są natomiast tutaj używane tagi tak jak w przypadku Ice.

Przykładowy IDL

```
namespace java genjava
namespace py genpython

struct BookOptional {
  1: string title,
  2: string author,
  3: optional i32 year,
}

struct BookNoOptional {
  1: string title,
  2: string author,
  3: i32 year,
}

service BookstoreService {
   bool addBookOptional(1:string title, 2: string author, 3: optional i32 year);
   bool addBookNoOptional(1:string title, 2: string author, 3: i32 year);
   bool addBookStructOptional(1: BookOptional book);
   bool addBookStructNoOptional(1: BookNoOptional book);
}
```

Uruchamiamy kod testujący:

```java
BookNoOptional bookNoOptional = new BookNoOptional();
bookNoOptional.setTitle("Harry Potter");
bookNoOptional.setAuthor("J.K. Rowling");
bookNoOptional.setYear(1997);

boolean res1 = client.addBookStructNoOptional(bookNoOptional);
System.out.println(res1);

BookOptional bookOptionalNotSet = new BookOptional();
bookNoOptional.setTitle("Harry Potter");
bookNoOptional.setAuthor("J.K. Rowling");

boolean res2 = client.addBookStructOptional(bookOptionalNotSet);
System.out.println(res2);

BookOptional bookOptionalSet = new BookOptional();
bookOptionalSet.setTitle("Harry Potter");
bookOptionalSet.setAuthor("J.K. Rowling");
bookOptionalSet.setYear(1997);

boolean res3 = client.addBookStructOptional(bookOptionalSet);
System.out.println(res3);
```

W Wiresharku analizujemy pakiety:

No.	Time	Source	Destination	Protocol	Length	Info
1331	24.395860	127.0.0.1	127.0.0.2	TCP	56	3036 → 9090 [SYN] Seq=0 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM
1332	24.395971	127.0.0.2	127.0.0.1	TCP	56	9090 → 3036 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM
1333	24.396016	127.0.0.1	127.0.0.2	TCP	44	3036 → 9090 [ACK] Seq=1 Ack=1 Win=2619648 Len=0
1334	24.439604	127.0.0.1	127.0.0.2	THRIFT	129	CALL addBookStructNoOptional
1335	24.439653	127.0.0.2	127.0.0.1	TCP	44	9090 → 3036 [ACK] Seq=1 Ack=86 Win=2619648 Len=0
1336	24.440520	127.0.0.2	127.0.0.1	THRIFT	134	EXCEPTION addBookStructNoOptional
1337	24.440564	127.0.0.1	127.0.0.2	TCP	44	3036 → 9090 [ACK] Seq=86 Ack=91 Win=2619648 Len=0
1351	24.950130	127.0.0.1	127.0.0.2	TCP	44	3036 → 9090 [RST, ACK] Seq=86 Ack=91 Win=0 Len=0
2708	85.266462	127.0.0.1	127.0.0.2	TCP	56	3050 → 9090 [SYN] Seq=0 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM
2709	85.266518	127.0.0.2	127.0.0.1	TCP	56	9090 → 3050 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM
2710	85.266595	127.0.0.1	127.0.0.2	TCP	44	3050 → 9090 [ACK] Seq=1 Ack=1 Win=2619648 Len=0
2711	85.302066	127.0.0.1	127.0.0.2	THRIFT	129	CALL addBookStructNoOptional
2712	85.302116	127.0.0.2	127.0.0.1	TCP	44	9090 → 3050 [ACK] Seq=1 Ack=86 Win=2619648 Len=0
2713	85.302312	127.0.0.2	127.0.0.1	THRIFT	134	EXCEPTION addBookStructNoOptional
2714	85.302351	127.0.0.1	127.0.0.2	TCP	44	3050 → 9090 [ACK] Seq=86 Ack=91 Win=2619648 Len=0
2740	85.800006	127.0.0.1	127.0.0.2	TCP	44	3050 → 9090 [RST, ACK] Seq=86 Ack=91 Win=0 Len=0
13815	574.994510	127.0.0.1	127.0.0.2	TCP	56	3148 → 9090 [SYN] Seq=0 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM
13816	574.994591	127.0.0.2	127.0.0.1	TCP	56	9090 → 3148 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM
13817	574.994620	127.0.0.1	127.0.0.2	TCP	44	3148 → 9090 [ACK] Seq=1 Ack=1 Win=2619648 Len=0
13818	575.024329	127.0.0.1	127.0.0.2	THRIFT	129	CALL addBookStructNoOptional
13819	575.024364	127.0.0.2	127.0.0.1	TCP	44	9090 → 3148 [ACK] Seq=1 Ack=86 Win=2619648 Len=0
13820	575.025043	127.0.0.2	127.0.0.1	THRIFT	84	REPLY addBookStructNoOptional
13821	575.025079	127.0.0.1	127.0.0.2	TCP	44	3148 → 9090 [ACK] Seq=86 Ack=41 Win=2619648 Len=0
13822	575.042421	127.0.0.1	127.0.0.2	THRIFT	82	CALL addBookStructOptional
13823	575.042455	127.0.0.2	127.0.0.1	TCP	44	9090 → 3148 [ACK] Seq=41 Ack=124 Win=2619648 Len=0
13824	575.043127	127.0.0.2	127.0.0.1	THRIFT	82	REPLY addBookStructOptional
13825	575.043160	127.0.0.1	127.0.0.2	TCP	44	3148 → 9090 [ACK] Seq=124 Ack=79 Win=2619648 Len=0
13826	575.049680	127.0.0.1	127.0.0.2	THRIFT	127	CALL addBookStructOptional
13827	575.049743	127.0.0.2	127.0.0.1	TCP	44	9090 → 3148 [ACK] Seq=79 Ack=207 Win=2619648 Len=0
13828	575.050265	127.0.0.2	127.0.0.1	THRIFT	82	REPLY addBookStructOptional
13829	575.050303	127.0.0.1	127.0.0.2	TCP	44	3148 → 9090 [ACK] Seq=207 Ack=117 Win=2619648 Len=0
13830	575.050645	127.0.0.1	127.0.0.2	TCP	44	3148 → 9090 [FIN, ACK] Seq=207 Ack=117 Win=2619648 Len=0
13831	575.050681	127.0.0.2	127.0.0.1	TCP	44	9090 → 3148 [ACK] Seq=117 Ack=208 Win=2619648 Len=0
13832	575.050769	127.0.0.2	127.0.0.1	TCP	44	9090 → 3148 [FIN, ACK] Seq=117 Ack=208 Win=2619648 Len=0
13833	575.050790	127.0.0.1	127.0.0.2	TCP	44	3148 → 9090 [ACK] Seq=208 Ack=118 Win=2619648 Len=0

1. Struktury
    - Podanie wszystkich argumentów do metody *addBookStructNoOptional*
      TCP Payload wynosi 85 bajtów.
      Thrift Protocol Call length wynosi 23 bajty.
    - Nie podanie opcjonalnego argumentu do metody *addBookStructOptional*
      TCP Payload wynosi 76 bajtów.
      Thrift Protocol Call length wynosi 21 bajty.
    - Podanie wszystkich argumentów do metody *addBookStructOptional*
      TCP Payload wynosi 83 bajtów.
      Thrift Protocol Call length wynosi 21 bajty.

3. **gRPC**
   W przypadku gRPC opcjonalne wartości w strukturach danych (messsages) definiowane są przy pomocy słowa kluczowego “optional” w proto3, który pozwala na definiowanie pól jako "opcjonalnych. Opcjonalne pola nie muszą być ustawione, co oznacza, że nie będą one serializowane i przesyłane w sieci, jeśli ich wartość nie została ustawiona.

   Przykładowy IDL:

```
message AddBookOptionalRequest {
  string title = 1;
  string author = 2;
  optional int32 year = 3;
}

message AddBookNoOptionalRequest {
  string title = 1;
  string author = 2;
  int32 year = 3;
}

message AddBookResponse {
  bool response = 1;
}

service BookstoreService  {
  rpc AddBookOptional(AddBookOptionalRequest) returns (AddBookResponse);
  rpc AddBookNoOptional(AddBookNoOptionalRequest) returns (AddBookResponse);
}
```

Uruchamiamy kod testujący:

```
BookstoreServiceProtos.AddBookNoOptionalRequest requestNoOptional = BookstoreServiceProtos.AddBookNoOptionalRequest.newBuilder()
        .setTitle("Harry Potter")
        .setAuthor("J.K. Rowling")
        .setYear(1997)
        .build();
BookstoreServiceProtos.AddBookResponse responseOptional = stub.addBookNoOptional(requestNoOptional);
System.out.println("Response received: " + responseOptional.getResponse());

BookstoreServiceProtos.AddBookOptionalRequest requestOptionalNotSet = BookstoreServiceProtos.AddBookOptionalRequest.newBuilder()
        .setTitle("Harry Potter")
        .setAuthor("J.K. Rowling")
        .build();

BookstoreServiceProtos.AddBookResponse responseOptionalNotSet = stub.addBookOptional(requestOptionalNotSet);
System.out.println("Response received: " + responseOptionalNotSet.getResponse());

BookstoreServiceProtos.AddBookOptionalRequest requestOptionalSet = BookstoreServiceProtos.AddBookOptionalRequest.newBuilder()
        .setTitle("Harry Potter")
        .setAuthor("J.K. Rowling")
        .setYear(1997)
        .build();

BookstoreServiceProtos.AddBookResponse responseOptionalSet = stub.addBookOptional(requestOptionalSet);
System.out.println("Response received: " + responseOptionalSet.getResponse());
```

W Wiresharku analizujemy pakiety:

729	15.908626	127.0.0.1	127.0.0.2	TCP	56	4204 → 9090 [SYN] Seq=0 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM
730	15.908730	127.0.0.2	127.0.0.1	TCP	56	9090 → 4204 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM
731	15.908777	127.0.0.1	127.0.0.2	TCP	44	4204 → 9090 [ACK] Seq=1 Ack=1 Win=2619648 Len=0
732	15.915801	127.0.0.2	127.0.0.1	HTTP2	90	SETTINGS[0], WINDOW_UPDATE[0]
733	15.915861	127.0.0.1	127.0.0.2	TCP	44	4204 → 9090 [ACK] Seq=1 Ack=47 Win=2619648 Len=0
734	15.931502	127.0.0.1	127.0.0.2	HTTP2	114	Magic, SETTINGS[0], WINDOW_UPDATE[0]
735	15.931561	127.0.0.2	127.0.0.1	TCP	44	9090 → 4204 [ACK] Seq=47 Ack=71 Win=2619648 Len=0
736	15.931668	127.0.0.2	127.0.0.1	HTTP2	53	SETTINGS[0]
737	15.931713	127.0.0.1	127.0.0.2	TCP	44	4204 → 9090 [ACK] Seq=71 Ack=56 Win=2619648 Len=0
738	15.963792	127.0.0.1	127.0.0.2	HTTP2	53	SETTINGS[0]
739	15.963823	127.0.0.2	127.0.0.1	TCP	44	9090 → 4204 [ACK] Seq=56 Ack=80 Win=2619648 Len=0
742	16.001206	127.0.0.1	127.0.0.2	GRPC	249	HEADERS[3]: POST /tutorial.BookstoreService/AddBookNoOptional, DATA[3] (GRPC) (PROTOBUF)
743	16.001259	127.0.0.2	127.0.0.1	TCP	44	9090 → 4204 [ACK] Seq=56 Ack=285 Win=2619392 Len=0
744	16.002817	127.0.0.2	127.0.0.1	HTTP2	61	PING[0]
745	16.002884	127.0.0.1	127.0.0.2	TCP	44	4204 → 9090 [ACK] Seq=285 Ack=73 Win=2619648 Len=0
746	16.004679	127.0.0.1	127.0.0.2	HTTP2	61	PING[0]
747	16.004721	127.0.0.2	127.0.0.1	TCP	44	9090 → 4204 [ACK] Seq=73 Ack=302 Win=2619392 Len=0
748	16.006926	127.0.0.2	127.0.0.1	GRPCHT…	184	HEADERS[3]: 200 OK, DATA[3] (GRPC) (PROTOBUF), HEADERS[3], WINDOW_UPDATE[0]
749	16.006979	127.0.0.1	127.0.0.2	TCP	44	4204 → 9090 [ACK] Seq=302 Ack=213 Win=2619392 Len=0
750	16.025171	127.0.0.1	127.0.0.2	HTTP2	61	PING[0]
751	16.025225	127.0.0.2	127.0.0.1	TCP	44	9090 → 4204 [ACK] Seq=213 Ack=319 Win=2619392 Len=0
752	16.025327	127.0.0.2	127.0.0.1	HTTP2	61	PING[0]
753	16.025409	127.0.0.1	127.0.0.2	TCP	44	4204 → 9090 [ACK] Seq=319 Ack=230 Win=2619392 Len=0
762	16.070701	127.0.0.1	127.0.0.2	GRPC	151	HEADERS[5]: POST /tutorial.BookstoreService/AddBookOptional, DATA[5] (GRPC) (PROTOBUF)
763	16.070731	127.0.0.2	127.0.0.1	TCP	44	9090 → 4204 [ACK] Seq=230 Ack=426 Win=2619136 Len=0
764	16.072021	127.0.0.2	127.0.0.1	GRPCHT…	95	HEADERS[5]: 200 OK, DATA[5] (GRPC) (PROTOBUF), HEADERS[5], WINDOW_UPDATE[0]
765	16.072072	127.0.0.1	127.0.0.2	TCP	44	4204 → 9090 [ACK] Seq=426 Ack=281 Win=2619392 Len=0
770	16.075141	127.0.0.1	127.0.0.2	GRPC	111	HEADERS[7]: POST /tutorial.BookstoreService/AddBookOptional, DATA[7] (GRPC) (PROTOBUF)
771	16.075176	127.0.0.2	127.0.0.1	TCP	44	9090 → 4204 [ACK] Seq=281 Ack=493 Win=2619136 Len=0
772	16.076364	127.0.0.2	127.0.0.1	GRPCHT…	95	HEADERS[7]: 200 OK, DATA[7] (GRPC) (PROTOBUF), HEADERS[7], WINDOW_UPDATE[0]
773	16.076424	127.0.0.1	127.0.0.2	TCP	44	4204 → 9090 [ACK] Seq=493 Ack=332 Win=2619392 Len=0
776	16.112628	127.0.0.2	127.0.0.1	HTTP2	61	PING[0]
777	16.112691	127.0.0.1	127.0.0.2	TCP	44	4204 → 9090 [ACK] Seq=493 Ack=349 Win=2619392 Len=0
823	16.677492	127.0.0.1	127.0.0.2	TCP	44	4204 → 9090 [RST, ACK] Seq=493 Ack=349 Win=0 Len=0

1. Wiadomości
    - Podanie wszystkich argumentów do metody *addBookStructNoOptional*
      TCP Payload wynosi 205 bajtów.
      GRPC Message Data wynosi 31 bajtów.
    - Nie podanie opcjonalnego argumentu do metody *addBookStructOptional*
      TCP Payload wynosi 105 bajtów.
      GRPC Message Data wynosi 28 bajtów.
    - Podanie wszystkich argumentów do metody *addBookStructOptional*
      TCP Payload wynosi 67 bajtów.
      GRPC Message Data wynosi 31 bajtów.

4. **Wnioski**
   Opcjonalne wartości w strukturach danych i argumentach wywołania middleware pozwalają na bardziej elastyczną komunikację między aplikacjami. Dzięki nim, można uniknąć przesyłania zbędnych informacji, co w przypadku Ice, jak i gRPC prowadzi do mniejszych rozmiarów pakietów sieciowych. W przypadku Thrift rozmiar pakietów sieciowych pozostaje taki sam niezależnie, czy parametry opcjonalne są ustawione, czy też nie.