sprawozdanie z lab. 4

Wstęp:

Do testowania aplikacji zostały użyte dwie wirtualne maszyny z linuxem połączone za pomocą serwisu hamachi.

Do analizy pakietów i ich zawartości użyto programu wireshark.

Zadanie 1:

Widok klienta:

```
$ ./bin/client1 127.0.0.1 5255
witam
Echo: witam
czesc
Echo: czesc
^C
```

Widok serwera:

```
$ ./bin/server1 5255
witam
czesc
```

Cała komunikacja została przedstawiona na poniższym zrzucie ekranu:

1	0.000000000	127.0.0.1	127.0.0.1	SCTP	106	INIT
2	0.000030961	127.0.0.1	127.0.0.1	SCTP	394	INIT_ACK
3	0.000046391	127.0.0.1	127.0.0.1	SCTP	342	COOKIE_ECHO
4	0.000071541	127.0.0.1	127.0.0.1	SCTP	50	COOKIE_ACK
5	1.986576689	25.33.152.95	25.33.152.95	SCTP	98	HEARTBEAT
6	1.986591832	25.33.152.95	25.33.152.95	SCTP	98	HEARTBEAT_ACK
7	2.914592498	25.33.152.95	25.33.152.95	SCTP	98	HEARTBEAT
8	2.914605275	10.0.2.15	10.0.2.15	SCTP	98	HEARTBEAT
9	2.914684511	25.33.152.95	25.33.152.95	SCTP	98	HEARTBEAT_ACK
10	2.914703408	10.0.2.15	10.0.2.15	SCTP	98	HEARTBEAT_ACK
11	3.682741995	10.0.2.15	10.0.2.15	SCTP	98	HEARTBEAT
12	3.682874294	10.0.2.15	10.0.2.15	SCTP	98	HEARTBEAT_ACK
13	18.430844687	127.0.0.1	127.0.0.1	SCTP	70	DATA
14	18.430861132	127.0.0.1	127.0.0.1	SCTP	62	SACK
15	18.430886417	127.0.0.1	127.0.0.1	SCTP	70	DATA
16	18.430890902	127.0.0.1	127.0.0.1	SCTP	62	SACK
17	33.378763269	25.33.152.95	25.33.152.95	SCTP	98	HEARTBEAT
18	33.378822433	25.33.152.95	25.33.152.95	SCTP	98	HEARTBEAT_ACK
19	35.426930756	10.0.2.15	10.0.2.15	SCTP	98	HEARTBEAT
20	35.426947986	10.0.2.15	10.0.2.15	SCTP	98	HEARTBEAT
21	35.426958039	25.33.152.95	25.33.152.95	SCTP	98	HEARTBEAT
22	35.427034820	10.0.2.15	10.0.2.15	SCTP	98	HEARTBEAT_ACK
23	35.427052737	10.0.2.15	10.0.2.15	SCTP	98	HEARTBEAT_ACK
24	35.427068040	25.33.152.95	25.33.152.95	SCTP	98	HEARTBEAT_ACK
25	36.982555285	127.0.0.1	127.0.0.1	SCTP	70	DATA
26	36.982586472	127.0.0.1	127.0.0.1	SCTP	86	SACK DATA
27	37.182689307	127.0.0.1	127.0.0.1	SCTP	62	SACK
28	40.705497618	127.0.0.1	127.0.0.1	SCTP	54	SHUTDOWN
29	40.705565877	127.0.0.1	127.0.0.1	SCTP	50	SHUTDOWN_ACK
30	40.705598098	127.0.0.1	127.0.0.1	SCTP	50	SHUTDOWN_COMPLETE

Jak widać 4 pierwsze wykryte pakiety należą do 4-way handshake.

Pakiet INIT wysłany przez klienta (port 43021):

```
▾ INIT chunk (Outbound streams: 10, inbound streams: 65535)
  ▸ Chunk type: INIT (1)
    Chunk flags: 0x00
    Chunk length: 60
    Initiate tag: 0x82b07c42
    Advertised receiver window credit (a_rwnd): 106496
    Number of outbound streams: 10
    Number of inbound streams: 65535
    Initial TSN: 2496907880
  ▸ IPv4 address parameter (Address: 127.0.0.1)
  ▸ IPv4 address parameter (Address: 10.0.2.15)
  ▸ IPv4 address parameter (Address: 25.33.152.95)
  ▸ Supported address types parameter (Supported types: IPv4)
  ▸ ECN parameter
  ▸ Forward TSN supported parameter
```

Numer TSN = 2493907880
Tag = 0x82b07c42

Pakiet INIT ACK od serwera (port 5255):

```
▾ INIT_ACK chunk (Outbound streams: 10, inbound streams: 10)
  ▸ Chunk type: INIT_ACK (2)
    Chunk flags: 0x00
    Chunk length: 348
    Initiate tag: 0xf4da2ff4
    Advertised receiver window credit (a_rwnd): 106496
    Number of outbound streams: 10
    Number of inbound streams: 10
    Initial TSN: 1303622300
  ▸ IPv4 address parameter (Address: 127.0.0.1)
  ▸ IPv4 address parameter (Address: 10.0.2.15)
  ▸ IPv4 address parameter (Address: 25.33.152.95)
  ▸ State cookie parameter (Cookie length: 292 bytes)
  ▸ ECN parameter
  ▸ Forward TSN supported parameter
```

Numer TSN = 1303622300
Tag = 0xf4da2ff4

Jak widać klient i serwer ustalają swój własny Initiate tag i Initial TSN.

Teraz w pakietach danych wysyłanych przez klienta znajduje się TSN ustalony wcześniej przez niego:

```
  Source port: 43021
  Destination port: 5255
  Verification tag: 0xf4da2ff4
  [Association index: 65535]
  Checksum: 0x00000000 [unverified]
  [Checksum Status: Unverified]
▸ DATA chunk(ordered, complete segment, TSN: 2496907880, SID: 0, SSN: 0, PPID: 0, payload length: 6 bytes)
```

Numer TSN = 2493907880
Tag = 0xf4da2ff4

Serwer odpowiada potwierdzeniem z tym samym numerem TSN:

```
Source port: 5255
Destination port: 43021
Verification tag: 0x82b07c42
[Association index: 65535]
Checksum: 0x00000000 [unverified]
[Checksum Status: Unverified]
SACK chunk (Cumulative TSN: 2496907880, a_rwnd: 106490, gaps: 0, duplicate TSNs: 0)
```

Numer TSN = 2493907880
Tag = 0x82b07c42

Natomiast jeśli to serwer wysyła pakiet z danymi to jego numer TSN i numer TSN pakietu potwierdzającego od klienta mają wartość ustaloną przez serwer.

```
Source port: 5255
Destination port: 43021
Verification tag: 0x82b07c42
[Association index: 65535]
Checksum: 0x00000000 [unverified]
[Checksum Status: Unverified]
▾ DATA chunk(ordered, complete segment, TSN: 1303622300, SID: 0, SSN: 0, PPID: 0, payload length: 6 bytes)
```

Numer TSN = 1303622300
Tag = 0x82b07c42

Potwierdzenie od klienta:

```
Source port: 43021
Destination port: 5255
Verification tag: 0xf4da2ff4
[Association index: 65535]
Checksum: 0x00000000 [unverified]
[Checksum Status: Unverified]
SACK chunk (Cumulative TSN: 1303622300, a_rwnd: 106490, gaps: 0, duplicate TSNs: 0)
```

Numer TSN = 1303622300
Tag = 0xf4da2ff4

Dodatkowo numer TSN zwiększa się o 1 po każdym użyciu go do komunikacji. Dobrze to widać na poniższym pakiecie, który równocześnie odpowiada i wysyła swoje dane, a który odbył się zaraz po powyższej komunikacji:

```
Source port: 5255
Destination port: 43021
Verification tag: 0x82b07c42
[Association index: 65535]
Checksum: 0x00000000 [unverified]
[Checksum Status: Unverified]
SACK chunk (Cumulative TSN: 2496907881, a_rwnd: 106496, gaps: 0, duplicate TSNs: 0)
DATA chunk(ordered, complete segment, TSN: 1303622301, SID: 0, SSN: 1, PPID: 0, payload length: 6 bytes)
```

Natomiast **Verification Tag zawsze zależy od tego kto wysyła dany pakiet**, niezależnie czy inicjując wymianę danych czy ją potwierdzając. Zatem serwer zawsze wkłada do swojego pakietu tag klienta i klient używa tagu ustalonego przez serwer. **Verification Tag jest stały,** nie zmienia się z następnymi pakietami.

Podsumowanie:

	Klient zainicjował wysyłanie danych	Serwer zainicjował wysyłanie danych	4-way handshake
Pakiet klienta	**DATA** Numer TSN = 2493907880+ Tag = 0xf4da2ff4	**SACK** Numer TSN = 1303622300+ Tag = 0xf4da2ff4	**INIT** Numer TSN = 2493907880 Tag = 0x82b07c42
Pakiet serwera	**SACK** Numer TSN = 2493907880+ Tag = 0x82b07c42	**DATA** Numer TSN = 1303622300+ Tag = 0x82b07c42	**INIT ACK** Numer TSN = 1303622300 Tag = 0xf4da2ff4

Zadanie 2

Wynik programu serwera:

```
czolg@czolg-VirtualBox:~/Documents/PUS/lab4/PUS-04-Protokol_SCTP-Linux/bin$ ./server2 5255
DATE SENT: 04/09/22
TIME SENT: 02:00:21
```

Wynik programu klienta (zrzut ekranu został zrobiony później niż poprzedni):

```
ID: 45
Association state: 4
Output sterams: 2
Input sterams: 2
Received 1: 04/09/22
Received 2: 13:51:32
```

Całość komunikacji pomiędzy serwerem a klientem (testowana na localhoscie):

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000000	127.0.0.1	127.0.0.1	SCTP	106	INIT
2	0.000030077	127.0.0.1	127.0.0.1	SCTP	394	INIT_ACK
3	0.000046044	127.0.0.1	127.0.0.1	SCTP	342	COOKIE_ECHO
4	0.000069160	127.0.0.1	127.0.0.1	SCTP	50	COOKIE_ACK
5	0.000110872	127.0.0.1	127.0.0.1	SCTP	318	DATA
6	0.000116010	127.0.0.1	127.0.0.1	SCTP	62	SACK
7	0.000125709	127.0.0.1	127.0.0.1	SCTP	318	DATA
8	0.000203276	127.0.0.1	127.0.0.1	SCTP	54	SHUTDOWN
9	0.000208764	127.0.0.1	127.0.0.1	SCTP	50	SHUTDOWN_ACK
10	0.000212531	127.0.0.1	127.0.0.1	SCTP	50	SHUTDOWN_COMPLETE

Liczba strumieni w pakiecie INIT:

```
INIT chunk (Outbound streams: 3, inbound streams: 4)
```

Liczba strumieni w pakiecie INIT ACK:

```
INIT_ACK chunk (Outbound streams: 2, inbound streams: 2)
```

Jak widać programu komunikują sobie swoje ilości możliwych strumieni przed rozpoczęciem transmisji danych.

Pierwszy pakiet DATA ma poniższe wartości:

```
Stream identifier: 0x0000
Stream sequence number: 0
```

Ponieważ dane zostały wysłane strumieniem 0 to identyfikator też jest 0.

Drugi pakiet DATA:

```
Stream identifier: 0x0001
Stream sequence number: 0
```

Numer strumienia, zgodnie z przewidywaniami, zmienił się na 1.

Ponieważ dane w żadnym z powyższych pakietów nie zostały podzielone na fragmenty to sequence number jest 0.

Program klienta po zmianie liczby strumieni akceptowanych na 1:

```
54      memset(&sctp_options, 0, sizeof(struct sctp_initmsg));
55      sctp_options.sinit_num_ostreams = 3;
56      sctp_options.sinit_max_instreams = 1; // zamiast 4
57      sctp_options.sinit_max_attempts = 5;
58      //sctp_options.sinit_max_init_timeo = ?;
```

Wynik serwera:

```
$ ./bin/server2 5255
DATE SENT: 04/09/22
sctp_sendmsg(): Invalid argument
```

Jak widać udało się wysłać tylko datę.

Sniffer pokazał tylko jeden pakiet DATA wymieniony, po którym zostało dokonane zamknięcie połączenia w wyniku błędy serwera:

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000000	127.0.0.1	127.0.0.1	SCTP	106	INIT
2	0.000030857	127.0.0.1	127.0.0.1	SCTP	394	INIT_ACK
3	0.000047011	127.0.0.1	127.0.0.1	SCTP	342	COOKIE_ECHO
4	0.000069889	127.0.0.1	127.0.0.1	SCTP	50	COOKIE_ACK
5	0.000193690	127.0.0.1	127.0.0.1	SCTP	318	DATA
6	0.000574419	127.0.0.1	127.0.0.1	SCTP	62	SACK
7	0.000582397	127.0.0.1	127.0.0.1	SCTP	54	SHUTDOWN
8	0.000587727	127.0.0.1	127.0.0.1	SCTP	50	SHUTDOWN_ACK
9	0.000591133	127.0.0.1	127.0.0.1	SCTP	50	SHUTDOWN_COMPLETE

Zadanie 3

serwer:

```
 /m/w/S/s/P/l/r/P/l/P/bin  main !2 ❯ ./server3 5255 0
Stream = 0
Id = 26
SSN = 0
TSN = 4068609922
received msg = 'CLIENT data'
Stream = 0
Id = 26
SSN = 1
TSN = 4068609923
received msg = 'CLIENT data'
Stream = 0
Id = 26
SSN = 2
TSN = 4068609924
received msg = 'CLIENT data'
Stream = 0
Id = 26
SSN = 3
TSN = 4068609925
received msg = 'CLIENT data'
Stream = 0
Id = 26
SSN = 4
TSN = 4068609926
received msg = 'CLIENT data'
Stream = 0
Id = 26
SSN = 5
TSN = 4068609927
```

klient:

```
 /m/w/S/s/P/l/r/P/l/P/bin  main !2 ❯ ./client3 127.0.0.1 5255
Stream = 0
Id = 25
SSN = 0
TSN = 2470081424
received msg = 'Sample Data from server'
Stream = 0
Id = 25
SSN = 1
TSN = 2470081425
received msg = 'Sample Data from server'
Stream = 0
Id = 25
SSN = 2
TSN = 2470081426
received msg = 'Sample Data from server'
Stream = 0
Id = 25
SSN = 3
TSN = 2470081427
received msg = 'Sample Data from server'
Stream = 0
Id = 25
SSN = 4
TSN = 2470081428
received msg = 'Sample Data from server'
```

Zastosowałem “wireShark” jako sniffer dla tego zadania by zaobserwować 4 way handshake plus wysyłane dane:

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000000	127.0.0.1	127.0.0.1	SCTP	106	INIT
2	0.000031861	127.0.0.1	127.0.0.1	SCTP	394	INIT_ACK
3	0.000051617	127.0.0.1	127.0.0.1	SCTP	614	COOKIE_ECHO DATA (TSN=0)
4	0.000081530	127.0.0.1	127.0.0.1	SCTP	66	COOKIE_ACK SACK (Ack=0, Arwnd=106240)
5	0.000213050	127.0.0.1	127.0.0.1	SCTP	318	DATA (TSN=0)
6	0.000283137	127.0.0.1	127.0.0.1	SCTP	62	SACK (Ack=0, Arwnd=106240)
7	0.000457855	127.0.0.1	127.0.0.1	SCTP	318	DATA (TSN=1)
8	0.000473481	127.0.0.1	127.0.0.1	SCTP	334	SACK (Ack=1, Arwnd=106496) DATA (TSN=1)
9	0.000636048	127.0.0.1	127.0.0.1	SCTP	334	SACK (Ack=1, Arwnd=106496) DATA (TSN=2)
10	0.000650136	127.0.0.1	127.0.0.1	SCTP	334	SACK (Ack=2, Arwnd=106496) DATA (TSN=2)
11	0.000809333	127.0.0.1	127.0.0.1	SCTP	334	SACK (Ack=2, Arwnd=106496) DATA (TSN=3)

W pakiecie INIT możemy zaobserwować ilość wchodzących i wychodzących streamów(tak samo w init_ack)
INIT:

```
Source port: 35732
Destination port: 5255
Verification tag: 0x00000000
[Association index: disabled (enable in preferences)]
Checksum: 0x00000000 [unverified]
[Checksum Status: Unverified]
INIT chunk (Outbound streams: 3, inbound streams: 3)
  Chunk type: INIT (1)
```

INIT_ACK:

```
Source port: 5255
Destination port: 35732
Verification tag: 0x584f5f74
[Association index: disabled (enable in preferences)]
Checksum: 0x00000000 [unverified]
[Checksum Status: Unverified]
INIT_ACK chunk (Outbound streams: 3, inbound streams: 3)
  Chunk type: INIT_ACK (2)
```

Różnice którą można zaobserwować w wyniku uruchamiania serwera z opcją 1 lub 0, jest taka że w przypadku opcji 0 id streamu będzie takie samo natomiast dla opcji 1 będzie ono nadawane w sekwencji 0 1 2 3 0 1 2 3 0 1 2…

Autorzy:

Przemysław Kożuch
Jarosław Kołodziej