

PSI LAB 2- Protokół TCP

13.12.2023, v1.0

Autorzy:

Igor Matynia, Andrii Gamalii, Wiktor Topolski, Bartłomiej Pełka

Treść zadania

Z 2 Komunikacja TCP

Napisać zestaw programów – klienta i serwera – komunikujących się poprzez TCP. Wykonać ćwiczenie w kolejnych inkrementalnych wariantach, rozszerzając kod z poprzedniej wersji.

Z 2.1

Napisać w języku C/Python klienta TCP, który wysyła złożoną strukturę danych, np. utworzoną w pamięci listę jednokierunkową lub drzewo binarne struktur zawierających (oprócz danych organizacyjnych) pewne dane dodatkowe, np. liczbę całkowitą 16-bitową, liczbę całkowitą 32-bitową oraz napis zmiennej i ograniczonej długości. Serwer napisany w Pythonie/C powinien te dane odebrać, dokonać poprawnego „odpakowania” tej struktury i wydrukować jej pola (być może w skróconej postaci, aby uniknąć nadmiaru wyświetlanych danych). Klient oraz serwer powinny być napisane w różnych językach.

Wskazówka: można wykorzystać moduły Pythona: struct i io.

Z 2.2

Zmodyfikować programy z zadania 2.1 tak, aby posługiwały się IPv6.

Z 2.3a

Zmodyfikować programy z zad. 2.1 w następujący sposób:

Klient powinien wysyłać do serwera strumień danych w pętli (danych powinno być przynajmniej kilkaset kB). Serwer powinien odbierać dane, ale między odczytami realizować sztuczne opóźnienie (np. za pomocą funkcji sleep()). W ten sposób symulowane będzie przeciążenie odbiorcy. Stos TCP będzie spowalniał nadawcę, aby uniknąć tracen danych. Zidentyfikować objawy tego zjawiska po stronie klienta (dodając pomiar i logowanie czasu, monitorując ruch sieciowy np. za pomocą narzędzi tcpdump lub Wireshark) i krótko przedstawić swoje wnioski poparte uzyskanymi obserwacjami i statystykami czasowymi. Przeprowadzić eksperymenty z różnymi rozmiarami bufora nadawczego po stronie klienta (np. 100 B, 1 kB, 10 kB).

Zadanie należy wykonać korzystając z kodu klienta i serwera napisanych w języku C.

Przesyłana struktura

Opis

Przesyланą strukturą jest lista jednokierunkowa, której elementy składają się po kolei z pól rating, price, title, content oraz pól związanych z organizacją struktury. Pole rating jest 16-bitową liczbą całkowitą, price 32-bitową liczbą całkowitą. Title jest ciągiem o stałej wielkości 255 znaków. Content natomiast jest ciągiem znaków o zmiennej długości.

Struktura węzła w C:

```
struct node {
    struct node *next;
    int id;
    int next_id;
    short rating;
    unsigned int price;
    char title[MAX_TEXT_LEN];
    unsigned int content_length;
    char* content;
};
```

Serializacja

Wysyłany bufor danych zawiera liczbę węzłów na początku (int - 4 bajty) i kolejno wszystkie zserializowane węzły listy.

Struktura zserializowanego węzła w C:

```
struct serialized_node{
    int id;
    int next_id;
    short rating;
    unsigned int price;
    char title[MAX_TEXT_LEN];
    unsigned int content_length;
};
```

Deserializacja

Najpierw węzły są zapisywane, a następnie łączone ze sobą w prawidłową listę poprzez referencje, za pomocą id węzłów.

Klient (python/c)

Serwer (c)

Konfiguracja testowa

Domyślnie serwer zostaje postawiony pod adresem localhost:::1 na porcie 8888. Ilość przesyłanych węzłów struktury to 10, a ilość danych w każdym z węzłów to 1000B. Ostatecznie przesyłanych jest 12764B danych. Każdy i-ty węzeł ma jako "title" ciąg znaków "Book by {i}" a zawartością content jest 1000 znaków "a".

Testy

Zadanie 2.1

Uruchomiony zostaje program c_serwer po czym uruchomiony zostaje klient napisany w pythonie.

```
Data sent
(domysl) imat@imat-IdeaPad-Gaming-3-15ARH05:~/psi/psi-lab-23z-z34/lab2tcp/zad2.1/python$ python3 py_client.py v4 localhost 8888 10 1000
Will connect to localhost : 8888
Sending 12764 bytes!
Data sent
```

Po stronie serwera ukazuje się komunikat:

```
(domysl) imat@imat-IdeaPad-Gaming-3-15ARH05:~/psi/psi-lab-23z-z34/lab2tcp/zad2.1/c$ ./c_server
Node: 0, rating: 0, price: 0, title: "Book by 0", content length: 1000
Node: 1, rating: 2, price: 3, title: "Book by 1", content length: 1000
Node: 2, rating: 4, price: 6, title: "Book by 2", content length: 1000
Node: 3, rating: 6, price: 9, title: "Book by 3", content length: 1000
Node: 4, rating: 8, price: 12, title: "Book by 4", content length: 1000
Node: 5, rating: 10, price: 15, title: "Book by 5", content length: 1000
Node: 6, rating: 12, price: 18, title: "Book by 6", content length: 1000
Node: 7, rating: 14, price: 21, title: "Book by 7", content length: 1000
Node: 8, rating: 16, price: 24, title: "Book by 8", content length: 1000
Node: 9, rating: 18, price: 27, title: "Book by 9", content length: 1000
./c_server: Ending connection
```

Dane zostały wysłane i odebrane pomyślnie.

Zadanie 2.2

Uruchomiony zostaje program c_serwer zadania 2.2, po czym uruchomiony zostaje klient napisany w pythonie z poniższymi argumentami wywołania.

```
(domysl) imat@imat-IdeaPad-Gaming-3-15ARH05:~/psi/psi-lab-23z-z34/lab2tcp/zad2.1/python$ python3 py_client.py v6 :::1 8888 10 1000
Will connect to :::1 : 8888
Sending 12764 bytes!
Data sent
```

W tym wypadku pierwszym argumentem wywołania było v6, pozwalające na obsługę IPv6. Po pomyślnym wystąpieniu, w logach serwera ukazuje się lista:

```
(domysl) imat@imat-IdeaPad-Gaming-3-15ARH05:~/psi/psi-lab-23z-z34/lab2tcp/zad2.2/c$ ./c_server
Socket port #8888
Node: 0, rating: 0, price: 0, title: "Book by 0", content length: 1000
Node: 1, rating: 2, price: 3, title: "Book by 1", content length: 1000
Node: 2, rating: 4, price: 6, title: "Book by 2", content length: 1000
Node: 3, rating: 6, price: 9, title: "Book by 3", content length: 1000
Node: 4, rating: 8, price: 12, title: "Book by 4", content length: 1000
Node: 5, rating: 10, price: 15, title: "Book by 5", content length: 1000
Node: 6, rating: 12, price: 18, title: "Book by 6", content length: 1000
Node: 7, rating: 14, price: 21, title: "Book by 7", content length: 1000
Node: 8, rating: 16, price: 24, title: "Book by 8", content length: 1000
Node: 9, rating: 18, price: 27, title: "Book by 9", content length: 1000
```

Dane zostały wysłane i odebrane pomyślnie.

Zmiany wprowadzone dla zadania 2.2

Klient python:

Dodany został argument wywołania pozwalający na wybór wersji protokołu ip. Wybranie wersji decyduje o tym czy przy inicjalizacji socketu użyjemy wartości AF_INET czy AF_INET6. Przy połączeniu w przypadku IPv6 do krotki adresu należy dodać 2 dodatkowe argumenty: flow info oraz scope id, w naszym wypadku równe 0.

```
if ip == "v4":
    with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as s:
        s.connect((host, port))
        s.sendall(buffer)
elif ip == "v6":
    with socket.socket(socket.AF_INET6, socket.SOCK_STREAM) as s:
        s.connect((host, port, 0, 0))
        s.sendall(buffer)
```

Serwer c:

Zmieniona została część kody odpowiedzialna za utworzenie gniazda. Użyte zostało AF_INET6 zamiast AF_INET.

```

int main(int argc, char **argv) {
    int sock, msgsock, length, rval, ListenQueueSize = 5, read_bytes = 0;
    struct sockaddr_in6 server;
    char buf[BSIZE], bufbuf[65356];
    linked_list_t ll;

    sock = socket( domain: AF_INET6, type: SOCK_STREAM, protocol: 0);
    if (sock < 0) bailout("opening stream socket");

    /* dowiaz adres IPv6 do gniazda */
    server.sin6_family = AF_INET6;
    // inet_pton(AF_INET6, DEF_ADDR, &(server.sin6_addr));
    server.sin6_addr = in6addr_any;
    server.sin6_port = htons( hostshort: DEF_PORT);
    if (bind( fd: sock, addr: (struct sockaddr *) &server, len: sizeof server) == -1)
        bailout("binding stream socket");

    /* wydrukuj na konsoli przydzielony port */
    length = sizeof(server);
    if (getsockname( fd: sock, addr: (struct sockaddr *) &server, len: &length) == -1)
        bailout("getting socket name");
    printf( format: "Socket port %#d\n", ntohs( netshort: server.sin6_port));
    // char str[INET6_ADDRSTRLEN];
    // inet_ntop(server.sin6_family, &server.sin6_addr, str, INET6_ADDRSTRLEN);
    // printf("Socket address: %s\n", str);
    /* zacznij przyjmować połączenia... */
    listen( fd: sock, n: ListenQueueSize);
}

```

Zadanie 2.3

Uruchomiony zostaje program c_serwer zadania 2.3, po czym uruchomiony zostaje klient napisany w c zadania 2.3 z argumentami wywołania "8888 666", co powoduje połączenie z serwerem nasłuchującym na porcie 8888 i przesłanie listy z 666 elementami (której wielkość to kilkaset kilobajtów)

Zmiany wprowadzone dla zadania 2.3

W kodzie klienta dane są teraz wysyłane porcjami po 100 (lub inną zdefiniowaną ilość) bajtów naraz:

```

char part_buffer[SEND_BUF_SIZE];
unsigned int sent = 0, to_send = 0;
while (sent < buffer_len) {
    to_send = buffer_len - sent > SEND_BUF_SIZE ? SEND_BUF_SIZE : buffer_len - sent;
    memcpy(part_buffer, __src: buffer_to_send + sent, to_send);
    if (write(sock, part_buffer, to_send) == -1) {
        bailout("writing on stream socket");
    }
    long long send_time = timeInMilliseconds();
    long long delta = send_time - start_time;
    printf(__format: "%d B sent after %lld ms\n", sent, delta);
    sent += to_send;
}

```

W kodzie serwera dodany został sleep(1) w pętli czytającej dane z gniazda:

```
63         do {
64             sleep(__seconds:1);
65             memset(buf, __c:0, __n:sizeof buf);
66             if ((rval = read(msgsock, buf, __nbytes:8192)) < 0) {
```

Rozmiar bufora również został znacznie zwiększony

```
c_server.c (/mnt/d/Dokumenty/PW/5 Semestr/PSI/lab-23z-z34/lab2tcp/zad2.3/c)  c_server.c (/mnt/d/Dokumenty/PW/5 Semestr/PSI/lab-23z-z34/lab2tcp/zad2.1/c)
10 #define BUFSIZE 1024 17 #define BUFSIZE 1024
11 #define BUFBUFSIZE 512 18 #define BUFBUFSIZE 5350
12 19 #define bailout(s) { perror(s); exit(1); }
13 20 #define notDone() TRUE
14 21
15 22 #define DEF_PORT "8888"
16 23
17 24 int moreWork(void) {
18 25     return 1;
19 26 }
20 27
21 28 char buf[BUFSIZE], bufbuf[BUFBUFSIZE];
22 29
23 30 int main(int argc, char **argv) {
24 31     int sock, msgsock, rval, listenQueueSize = 5, read_bytes = 0;
25 32     socklen_t length;
26 33     struct addrinfo *bindto_address;
27 34     struct addrinfo hints;
28 35     struct sockaddr_in server;
29 36     char buf[BUFSIZE], bufbuf[5350];
30 37     linked_list_t ll;
```

Wyniki zadania 2.3

```
c_server x c_client x
...
1302200 B sent after 65 ms
1302300 B sent after 65 ms
1302400 B sent after 65 ms
1302500 B sent after 65 ms
1302600 B sent after 65 ms
1302700 B sent after 65 ms
1302800 B sent after 65 ms
1302900 B sent after 65 ms
1303000 B sent after 65 ms

Process finished with exit code 0
```

Klientowi przy każdej testowanej wielkości wysyłanego pakietu udaje się od razu wysłać całą listę w bardzo krótkim czasie.

Z perspektywy Wiresharka przesłane pakiety wyglądają tak:

tcp port == 8888						
No.	Time	Source	Destination	Protocol	Length	Info
37957	586.842458480	127.0.0.1	127.0.0.1	TCP	166	37588 → 8888 [PSH, ACK] Seq=1201 Ack=1 Win=32280 Len=100 TSval=3201137089 TSecr=3201137089 [T...
37958	586.842459235	127.0.0.1	127.0.0.1	TCP	66	8888 → 37588 [ACK] Seq=1 Ack=1301 Win=32280 Len=0 TSval=3201137089 TSecr=3201137089
37959	586.842462321	127.0.0.1	127.0.0.1	TCP	166	EXT
37960	586.842463173	127.0.0.1	127.0.0.1	TCP	66	8888 → 37588 [ACK] Seq=1 Ack=1401 Win=32280 Len=0 TSval=3201137089 TSecr=3201137089
37961	586.842466326	127.0.0.1	127.0.0.1	TCP	166	37588 → 8888 [PSH, ACK] Seq=1401 Ack=1 Win=32280 Len=100 TSval=3201137089 TSecr=3201137089
37962	586.842467209	127.0.0.1	127.0.0.1	TCP	66	8888 → 37588 [ACK] Seq=1 Ack=1501 Win=32280 Len=0 TSval=3201137089 TSecr=3201137089
37963	586.842470270	127.0.0.1	127.0.0.1	TCP	166	37588 → 8888 [PSH, ACK] Seq=1501 Ack=1 Win=32280 Len=100 TSval=3201137089 TSecr=3201137089
37964	586.842471197	127.0.0.1	127.0.0.1	TCP	66	8888 → 37588 [ACK] Seq=1 Ack=1601 Win=32280 Len=0 TSval=3201137089 TSecr=3201137089
37965	586.842474210	127.0.0.1	127.0.0.1	TCP	166	37588 → 8888 [PSH, ACK] Seq=1601 Ack=1 Win=32280 Len=100 TSval=3201137089 TSecr=3201137089
37966	586.842683727	127.0.0.1	127.0.0.1	TCP	16766	37588 → 8888 [ACK] Seq=1701 Ack=1 Win=32280 Len=16640 TSval=3201137089 TSecr=3201137089
37967	586.842685194	127.0.0.1	127.0.0.1	TCP	66	8888 → 37588 [ACK] Seq=1 Ack=18341 Win=32280 Len=0 TSval=3201137089 TSecr=3201137089
37968	586.842687610	127.0.0.1	127.0.0.1	TCP	126	37588 → 8888 [PSH, ACK] Seq=18341 Ack=1 Win=32280 Len=60 TSval=3201137089 TSecr=3201137089
37969	586.842692120	127.0.0.1	127.0.0.1	TCP	16766	37588 → 8888 [ACK] Seq=18401 Ack=1 Win=32280 Len=16640 TSval=3201137089 TSecr=3201137089 [TCP...
37970	586.842908886	127.0.0.1	127.0.0.1	TCP	66	8888 → 37588 [ACK] Seq=1 Ack=35041 Win=32280 Len=0 TSval=3201137089 TSecr=3201137089
37971	586.842902230	127.0.0.1	127.0.0.1	TCP	126	37588 → 8888 [PSH, ACK] Seq=35041 Ack=1 Win=32280 Len=60 TSval=3201137089 TSecr=3201137089 [T...
37972	586.843096636	127.0.0.1	127.0.0.1	TCP	16766	37588 → 8888 [ACK] Seq=35101 Ack=1 Win=32280 Len=16640 TSval=3201137089 TSecr=3201137089 [TCP...
37973	586.843097686	127.0.0.1	127.0.0.1	TCP	66	8888 → 37588 [ACK] Seq=1 Ack=51741 Win=32280 Len=0 TSval=3201137089 TSecr=3201137089
37974	586.843099951	127.0.0.1	127.0.0.1	TCP	126	37588 → 8888 [PSH, ACK] Seq=51741 Ack=1 Win=32280 Len=60 TSval=3201137089 TSecr=3201137089 [T...
37975	586.843277297	127.0.0.1	127.0.0.1	TCP	16766	37588 → 8888 [ACK] Seq=51801 Ack=1 Win=32280 Len=16640 TSval=3201137089 TSecr=3201137089 [TCP...
37976	586.843278584	127.0.0.1	127.0.0.1	TCP	66	8888 → 37588 [ACK] Seq=1 Ack=68441 Win=32280 Len=0 TSval=3201137089 TSecr=3201137089
37977	586.843280571	127.0.0.1	127.0.0.1	TCP	126	37588 → 8888 [PSH, ACK] Seq=68441 Ack=1 Win=32280 Len=60 TSval=3201137089 TSecr=3201137089 [T...
37978	586.843483548	127.0.0.1	127.0.0.1	TCP	16766	37588 → 8888 [ACK] Seq=68501 Ack=1 Win=32280 Len=16640 TSval=3201137089 TSecr=3201137089 [TCP...
37979	586.843484777	127.0.0.1	127.0.0.1	TCP	66	8888 → 37588 [ACK] Seq=1 Ack=85141 Win=32280 Len=0 TSval=3201137089 TSecr=3201137089
37980	586.843486916	127.0.0.1	127.0.0.1	TCP	126	37588 → 8888 [PSH, ACK] Seq=85141 Ack=1 Win=32280 Len=60 TSval=3201137089 TSecr=3201137089 [T...
37981	586.843785718	127.0.0.1	127.0.0.1	TCP	16766	37588 → 8888 [ACK] Seq=85201 Ack=1 Win=32280 Len=16640 TSval=3201137089 TSecr=3201137089 [TCP...
37982	586.848714784	127.0.0.1	127.0.0.1	TCP	16766	37588 → 8888 [ACK] Seq=101841 Ack=1 Win=32280 Len=16640 TSval=3201137089 TSecr=3201137089 [T...
37983	586.848724436	127.0.0.1	127.0.0.1	TCP	78	8888 → 37588 [ACK] Seq=1 Ack=101841 Win=32280 Len=0 TSval=3201137089 TSecr=3201137089 SLE=852...
Frame 119: 74 bytes on wire (592 bits), 74 bytes captured (592 bits) on interface lo (00:00:00:00:00:00), Dst: Xerox.00:00:00:00:00:00 Ethernet II, Src: Xerox.00:00:00:00:00:00, Dst: Xerox.00:00:00:00:00:00 Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1 Transmission Control Protocol, Src Port: 42996, Dst Port: 8888, Seq: 0, Len: 0 Source Port: 42996 Destination Port: 8888 [Stream index: 1] Conversation complete: Complete. WITH DATA (3111)						

No.	Time	Source	Destination	Protocol	Length	Info			
44566	846.938839541	127.0.0.1	127.0.0.1	TCP	16766	37588 → 8888 [ACK] Seq=334801 Ack=1 Win=32280 Len=16640 TSval=3201390286 TSecr=3201390286 [T...			
44567	846.938852549	127.0.0.1	127.0.0.1	TCP	16766	TCP Window Full() 37588 → 8888 [ACK] Seq=351441 Ack=1 Win=32280 Len=16640 TSval=3201390286 T...			
44568	846.938871752	127.0.0.1	127.0.0.1	TCP	16766	TCP Window Full() TCP Window Full() 37588 → 8888 [ACK] Seq=351441 Ack=1 Win=32280 Len=16640 TS...			
44569	846.938833009	127.0.0.1	127.0.0.1	TCP	78	8888 → 37588 [ACK] Seq=1 Ack=308081 Win=32280 Len=0 TSval=3201390306 TSecr=3201390306 SLE=351...			
44570	846.938865259	127.0.0.1	127.0.0.1	TCP	16766	37588 → 8888 [ACK] Seq=368801 Ack=1 Win=32280 Len=16640 TSval=3201390306 TSecr=3201390306 [T...			
44571	846.939712162	127.0.0.1	127.0.0.1	TCP	66	8888 → 37588 [ACK] Seq=1 Ack=384721 Win=32280 Len=0 TSval=3201390348 TSecr=3201390348			
44572	846.939741361	127.0.0.1	127.0.0.1	TCP	1340	TCP Window Full() 37588 → 8888 [PSH, ACK] Seq=384721 Ack=1 Win=32280 Len=1280 TSval=320139037...			
44573	846.939759620	127.0.0.1	127.0.0.1	TCP	66	TCP Window Full() 8888 → 37588 [ACK] Seq=1 Ack=386601 Win=0 Len=0 TSval=3201390376 TSecr=32013...			
44574	846.939761629	127.0.0.1	127.0.0.1	TCP	66	TCP Keep-Alive() 37588 → 8888 [ACK] Seq=386601 Ack=1 Win=32280 Len=0 TSval=3201390785 TSecr=3...			
44575	846.939778849	127.0.0.1	127.0.0.1	TCP	66	TCP Keep-Alive() 8888 → 37588 [ACK] Seq=386600 Ack=1 Win=32280 Len=0 TSval=3201390785 TSecr=32013...			
44582	846.939790999	127.0.0.1	127.0.0.1	TCP	66	TCP Keep-Alive() 37588 → 8888 [ACK] Seq=386600 Ack=1 Win=32280 Len=0 TSval=3201391217 TSecr=3...			
44591	846.9406763755	127.0.0.1	127.0.0.1	TCP	66	TCP Keep-Alive() 37588 → 8888 [ACK] Seq=386600 Ack=1 Win=32280 Len=0 TSval=3201392114 TSecr=3...			
44592	846.940709369	127.0.0.1	127.0.0.1	TCP	66	TCP Keep-Alive() 8888 → 37588 [ACK] Seq=1 Ack=386601 Win=0 Len=0 TSval=3201392144 TSecr=32013...			
44593	846.940734333	127.0.0.1	127.0.0.1	TCP	66	TCP Keep-Alive() 37588 → 8888 [ACK] Seq=386600 Ack=1 Win=32280 Len=0 TSval=3201392144 TSecr=32013...			
44594	846.9407360197	127.0.0.1	127.0.0.1	TCP	66	TCP Keep-Alive() 8888 → 37588 [ACK] Seq=1 Ack=386601 Win=0 Len=0 TSval=3201392341 TSecr=32013...			
44602	846.926775477	127.0.0.1	127.0.0.1	TCP	66	TCP Keep-Alive() 37588 → 8888 [ACK] Seq=128384 Ack=1 Win=32280 Len=0 TSval=3201390274 TSecr=3...			
44603	846.926780550	127.0.0.1	127.0.0.1	TCP	66	TCP Keep-Alive() 8888 → 37588 [ACK] Seq=386600 Ack=1 Win=32280 Len=0 TSval=3201390274 TSecr=3...			
44639	847.050709059	127.0.0.1	127.0.0.1	TCP	66	TCP Keep-Alive() 37588 → 8888 [ACK] Seq=386600 Ack=1 Win=32280 Len=0 TSval=3201391708 TSecr=3...			
44640	847.050710484	127.0.0.1	127.0.0.1	TCP	66	TCP Keep-Alive() 8888 → 37588 [ACK] Seq=1 Ack=386601 Win=0 Len=0 TSval=3201391708 TSecr=32013...			
44699	854.217168359	127.0.0.1	127.0.0.1	TCP	66	TCP Keep-Alive() 37588 → 8888 [ACK] Seq=386600 Ack=1 Win=32280 Len=0 TSval=3201404405 TSecr=3...			
44700	854.217169111	127.0.0.1	127.0.0.1	TCP	66	TCP Keep-Alive() 8888 → 37588 [ACK] Seq=1 Ack=386601 Win=0 Len=0 TSval=3201404405 TSecr=32013...			
44741	859.337743017	127.0.0.1	127.0.0.1	TCP	66	TCP Keep-Alive() 37588 → 8888 [ACK] Seq=128384 Ack=1 Win=32280 Len=0 TSval=3201404565 TSecr=3...			
44742	859.337743932	127.0.0.1	127.0.0.1	TCP	66	TCP Keep-Alive() 8888 → 37588 [ACK] Seq=1 Ack=386601 Win=0 Len=0 TSval=3201404565 TSecr=32013...			
44795	865.994755838	127.0.0.1	127.0.0.1	TCP	66	TCP Keep-Alive() 41576 → 8888 [ACK] Seq=127880 Ack=1 Win=32280 Len=0 TSval=3201410242 TSecr=3...			
44800	865.994756838	127.0.0.1	127.0.0.1	TCP	66	TCP Keep-Alive() 8888 → 37588 [ACK] Seq=386600 Ack=1 Win=32280 Len=0 TSval=3201410242 TSecr=3...			
44813	868.041804099	127.0.0.1	127.0.0.1	TCP	66	TCP Keep-Alive() 37588 → 8888 [ACK] Seq=386600 Ack=1 Win=32280 Len=0 TSval=3201410289 TSecr=3...			
44814	868.041806101	127.0.0.1	127.0.0.1	TCP	66	TCP Keep-Alive() 8888 → 37588 [ACK] Seq=1 Ack=386601 Win=0 Len=0 TSval=3201410289 TSecr=32013...			

Występuje sklejanie pakietów tcp oraz ich ewentualna retransmisja.
 Serwer ostrzega że bufor mu się skończył, a klient już dawno przestał nadawać