Stream Control Transmission Protocol (SCTP)

,

RFC 3286 [Ong and Yoakum 2002].

RFC 4960 September 2007 (http://tools.ietf.org/html/rfc4960)

(http://www.iana.org/assignments/sctp-parameters/sctp-parameters.xhtml)

R. R. Stewart, Q. Xie, „Stream Control Transmission Protocol (STCP) – A reference Guide, Addison Wesley, 2001

Podstawowe własności protokołu SCTP

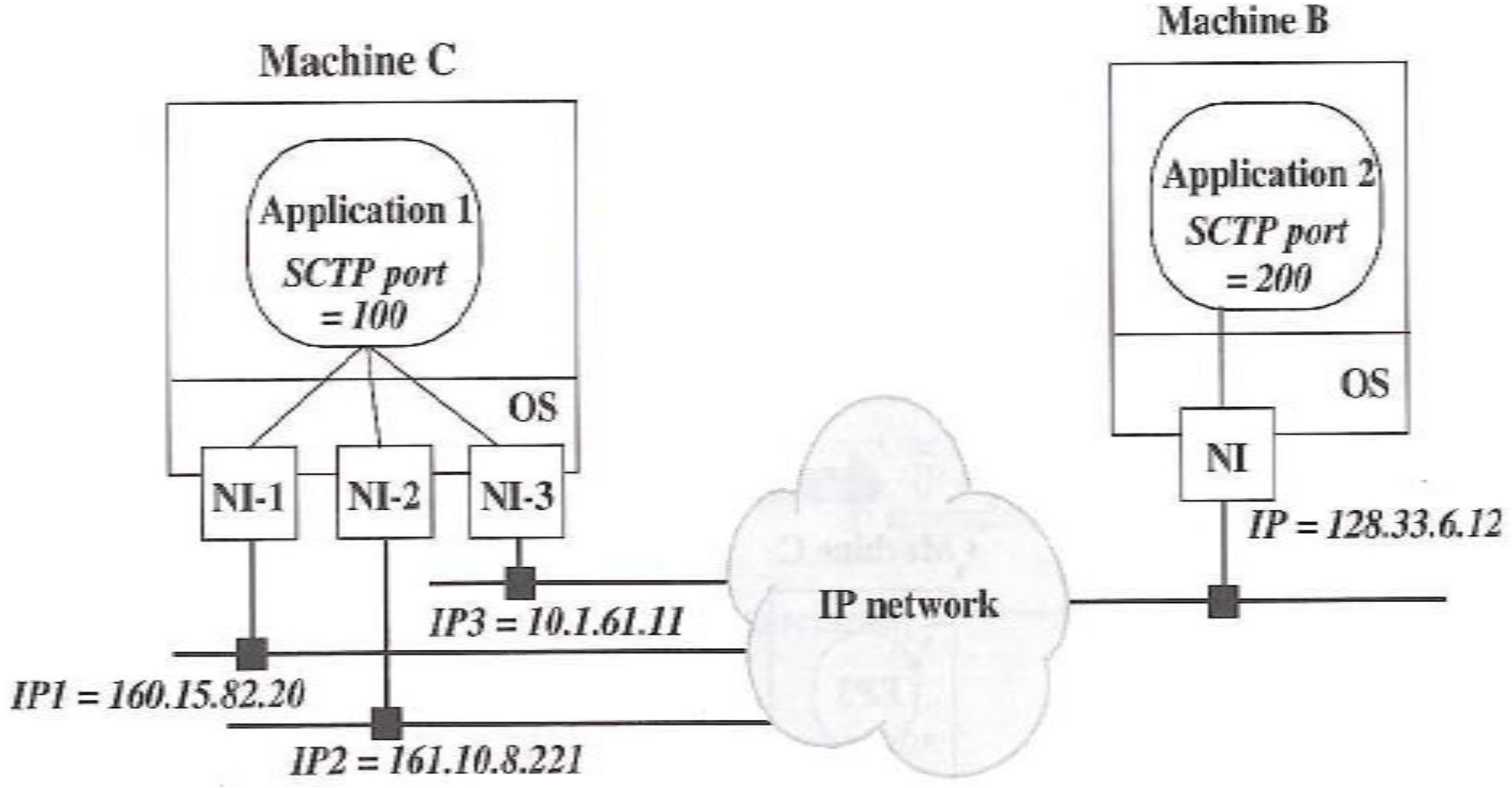
- SCTP jest protokołem obsługującym tylko adresy jednostkowe (unicast) i wspiera komunikację pomiędzy dokładnie dwoma systemami końcowymi, ale systemy końcowe mogą być reprezentowane przez wiele adresów IP.
- Protokół SCTP zapewnia transmisję niezawodną, wykrywanie danych utraconych, przemieszanych, powielonych i uszkodzonych oraz retransmisję danych w razie potrzeby. Asocjacja SCTP jest dwukierunkowa.
- SCTP jest zorientowany na wiadomości i wspiera zachowanie granic poszczególnych wiadomości po stronie odbiorczej. Inaczej niż protokół TCP, który jest zorientowany na transmisję strumienia bajtów i nie zachowuje żadnej struktury w transmitowanym strumieniu.
- Protokół SCTP jest adaptacyjny, podobnie jak TCP i będzie dostosowywać szybkość transmisji danych do obciążenia sieci. Szybkość strumienia danych TCP i SCTP na tej samej ścieżce powinna być podobna.

Asocjacja SCTP

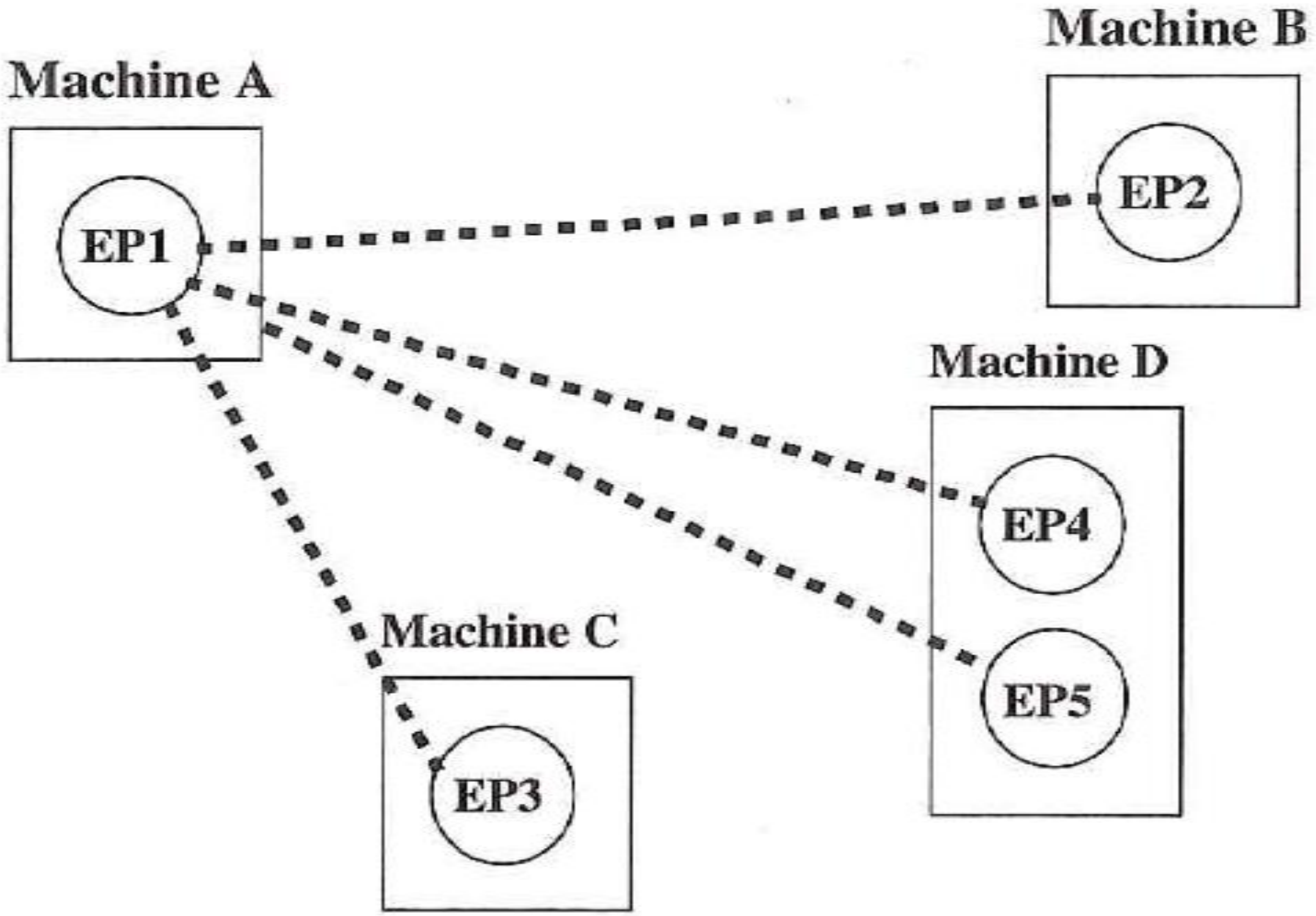
- Dla protokołu SCTP stosuje się pojęcie „asocjacja” zamiast „połączenie”, które jest stosowane do określenia wymiany danych pomiędzy dwoma adresami IP. Asocjacja nawiązuje do komunikacji pomiędzy dwoma systemami końcowymi, która może odbywać pomiędzy więcej niż dwoma adresami IP, gdy systemy końcowe posiadają więcej interfejsów (adresów) sieciowych.
- Dla pojedynczej asocjacji może istnieć kilka niezależnych strumieni danych. Zagubienie danych w jednej sesji nie blokuje dostarczania danych w innych sesjach.
- Na jednym gnieździe można obsługiwać kilka asocjacji (np.dla TCP można obsługiwać tylko jedno połączenie)

Asocjacja SCTP

[port źródłowy, tablica adresów źródłowych, port docelowy, tablica adresów docelowych]



Jedna lub wiele asocjacji na jednym gnieździe



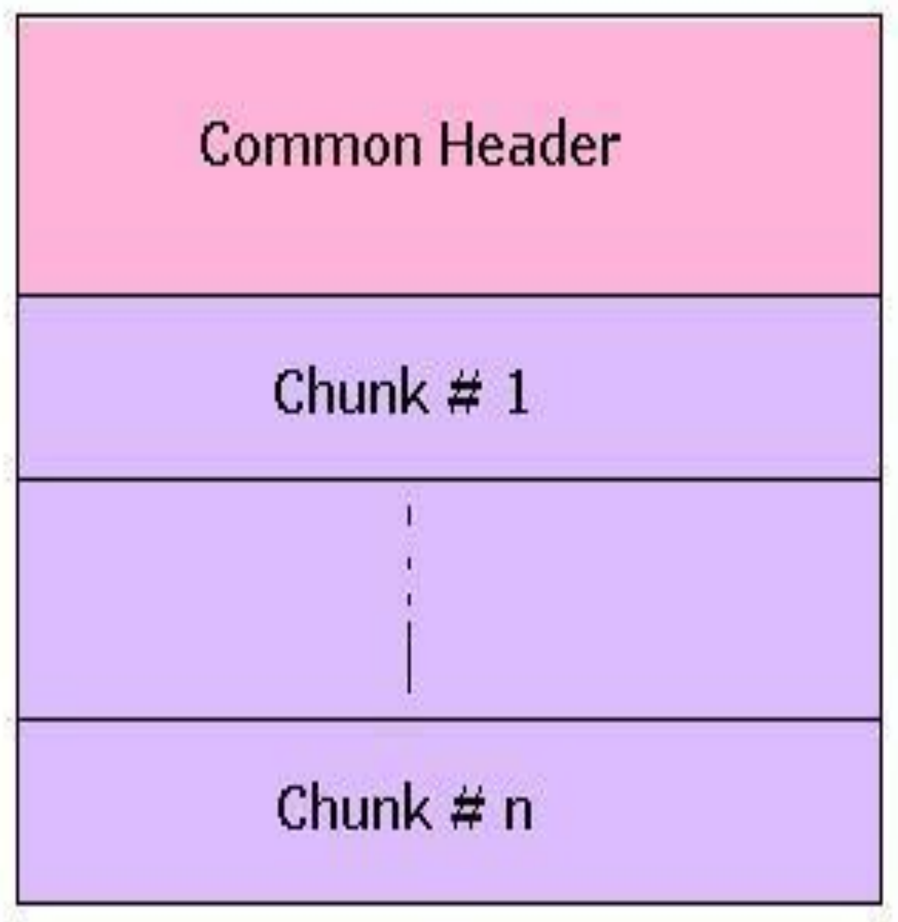
Protokół SCTP oferuje następujące usługi dla aplikacji:

- Bezbłędną transmisję danych z potwierdzeniami i eliminacją danych zduplikowanych.
- Fragmentację strumienia danych dostosowaną do PMTU (MTU dla ścieżki).
- Sekwencyjne dostarczanie komunikatów aplikacji za pomocą wielu strumieni datagramów, z opcją dostarczania komunikatów w takiej kolejności w jakiej są dostarczane do odbiorcy.
- Możliwość łączenia wielu komunikatów od aplikacji w jednym datagramie SCTP.
- Odporność na uszkodzenia na ścieżce danych w przypadku systemów końcowych wyposażonych w więcej niż jedną kartę sieciową (multi-homing)

SCTP [RFC4960]

```
           +--------------------------+
           |      User Messages       |
           +--------------------------+
 SCTP user        ^  |
==================|==|=======================================
                  |  v (1)
       +------------------+    +--------------------+
       | SCTP DATA Chunks |    |SCTP Control Chunks |
       +------------------+    +--------------------+
                  ^  |             ^  |
                  |  v (2)         |  v (2)
               +--------------------------+
               |      SCTP packets        |
               +--------------------------+
 SCTP                       ^  |
===========================|==|===========================
                           |  v
       Connectionless Packet Transfer Service (e.g., IP)
```

SCTP packet format



SCTP Common Header format

Source port no.	Destination port no.
Verification tag	
Checksum	

SCTP Chunk Field format (format bloku np. danych w konwencji Type Lenght Value)

Chunk 1 type	Chunk 1 flags	Chunk 1 length
Chunk 1 data		

SCTP packet format

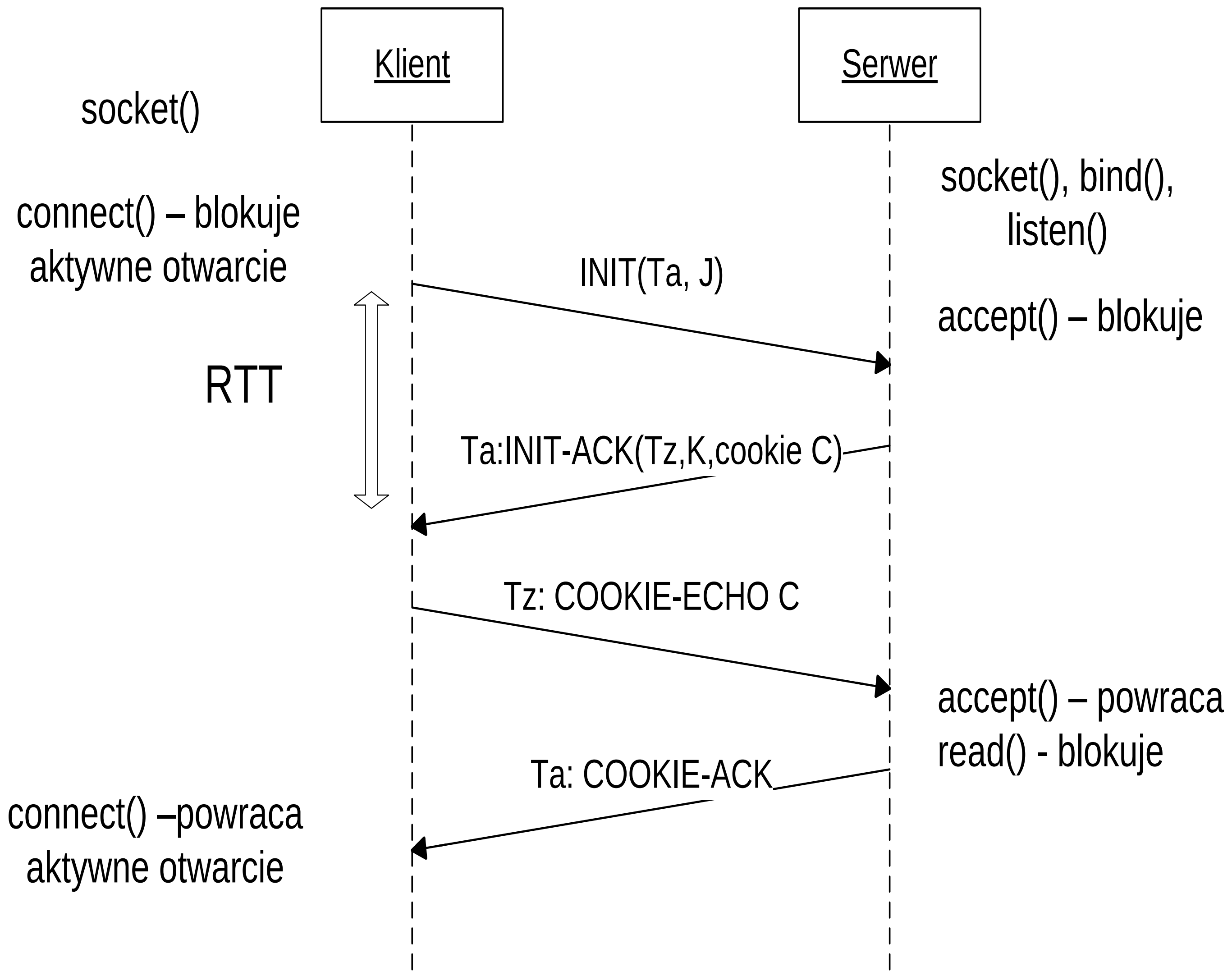
Bits	Bits 0 - 7	8 - 15	16 - 23	24 - 31
+0	Source port		Destination port	
32	Verification tag			
64	Checksum			
96	Chunk 1 type	Chunk 1 flags	Chunk 1 length	
128	Chunk 1 data			
…	…			
…	Chunk N type	Chunk N flags	Chunk N length	
…	Chunk N data			

SCTP Data chunk

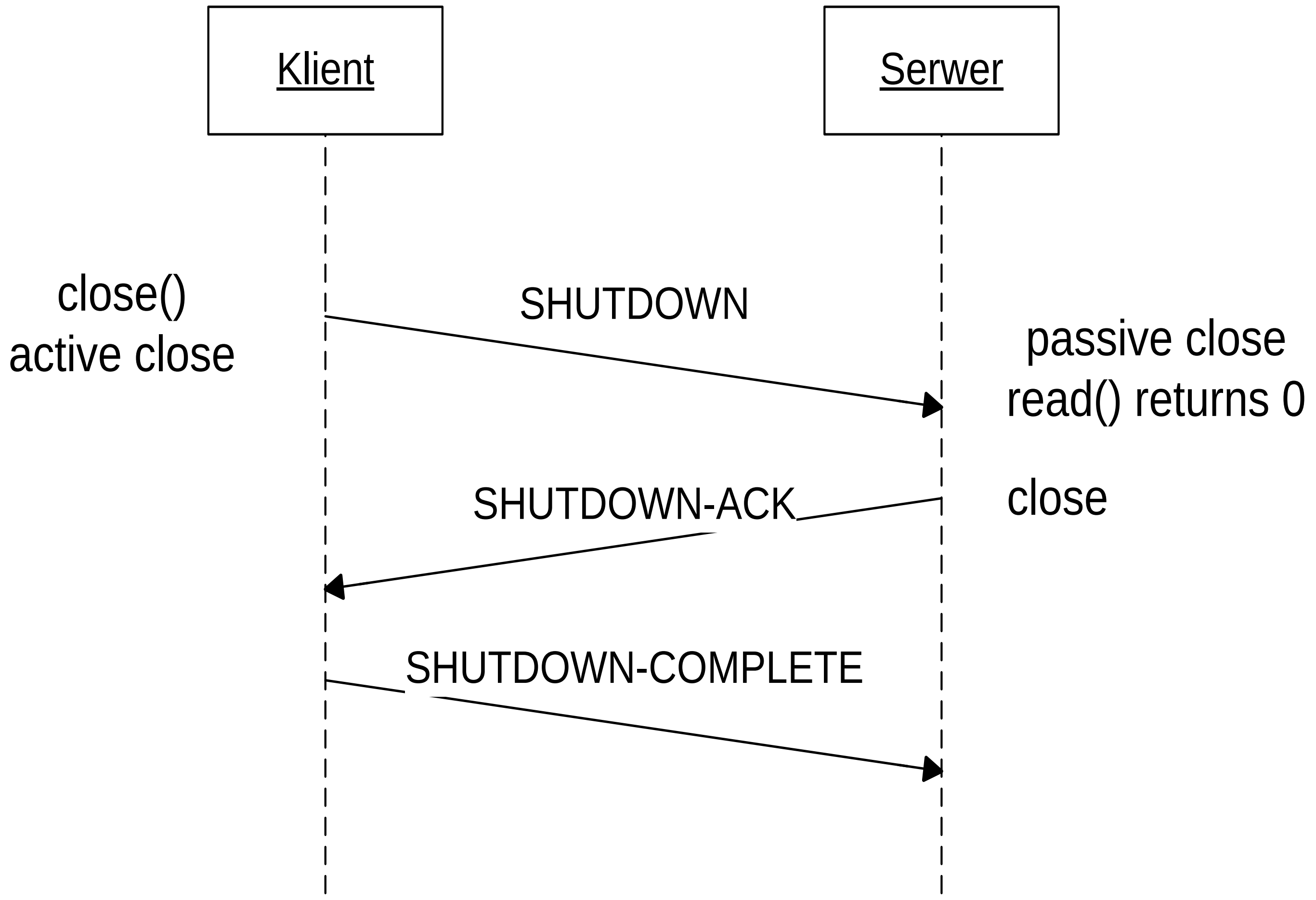
```
0                   1                   2                   3
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
| Type = 0     | Reserved|U|B|E|            Length              |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                              TSN                              |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|   Stream Identifier S         |    Stream Sequence Number n   |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                  Payload Protocol Identifier                  |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
\                                                               \
/                User Data (seq n of Stream S)                  /
\                                                               \
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+

Flags: U - unordered, B – begin, E - end
```

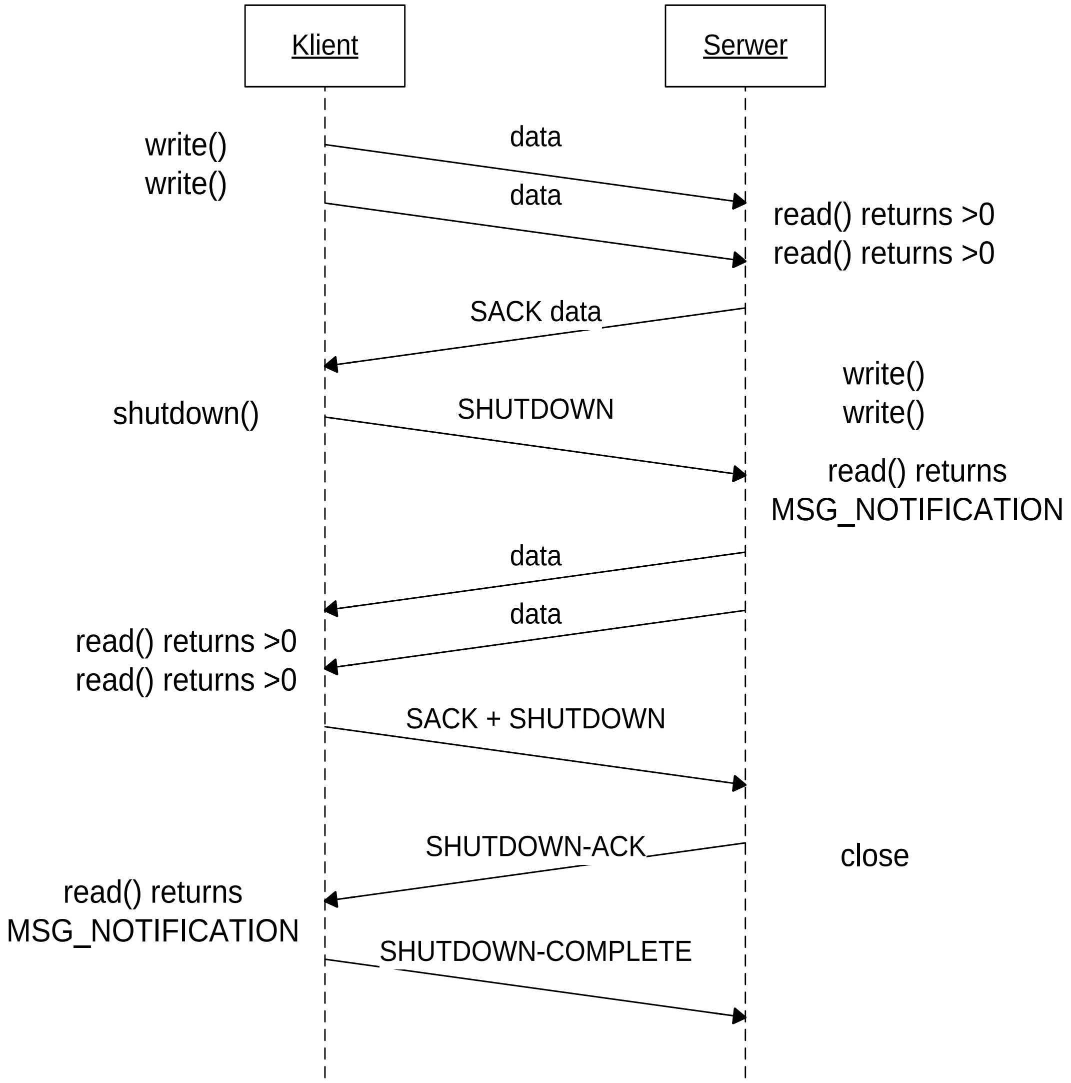
SCTP – connection setup


Klient
Serwer
socket()
connect() – blokuje aktywne otwarcie
socket(), bind(), listen()
INIT(Ta, J)
accept() – blokuje
RTT
Ta:INIT-ACK(Tz,K,cookie C)
Tz: COOKIE-ECHO C
accept() – powraca
read() - blokuje
Ta: COOKIE-ACK
connect() –powraca aktywne otwarcie

SCTP – connection tear down - close()

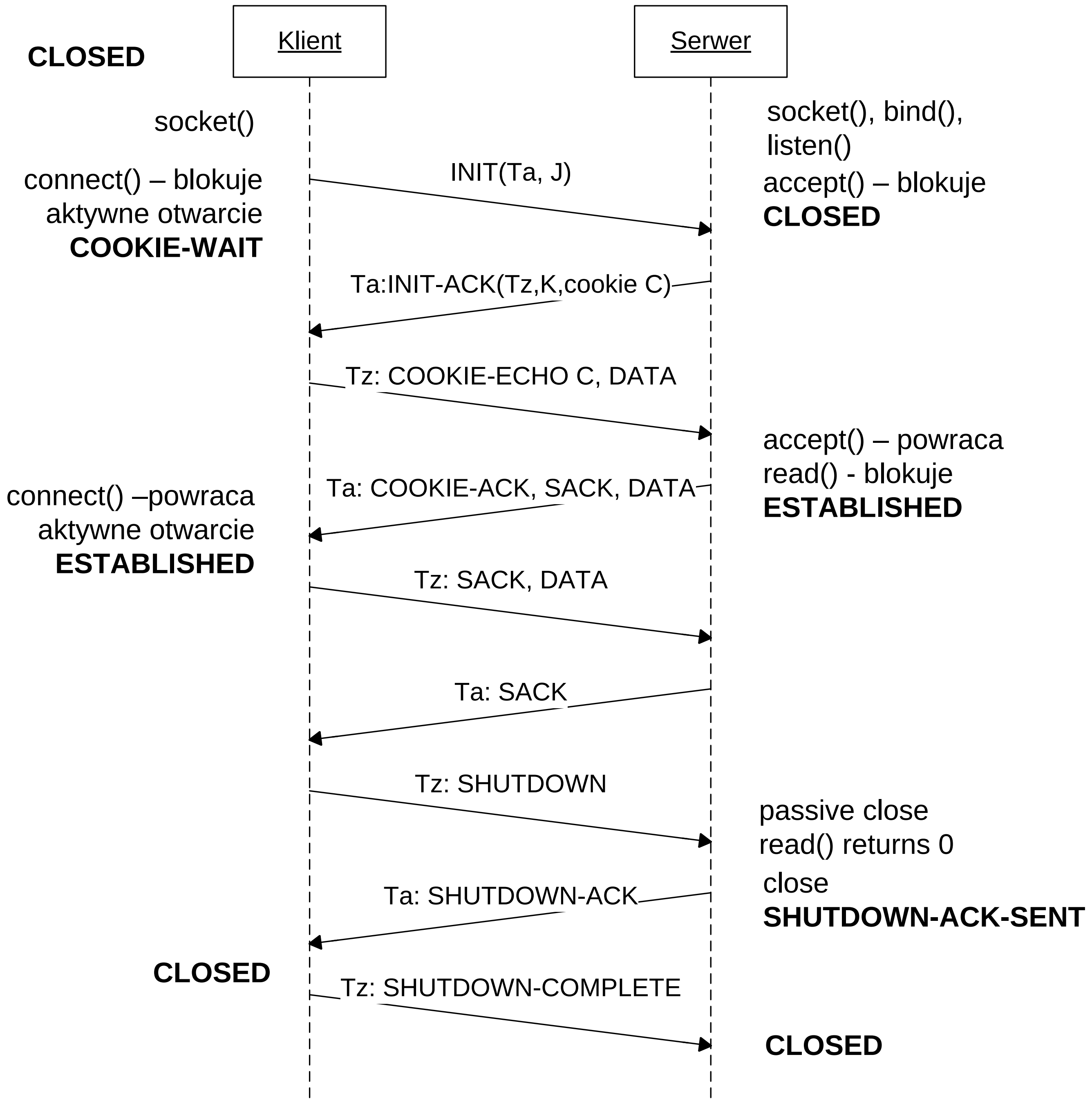


SCTP – connection tear down – shutdown()

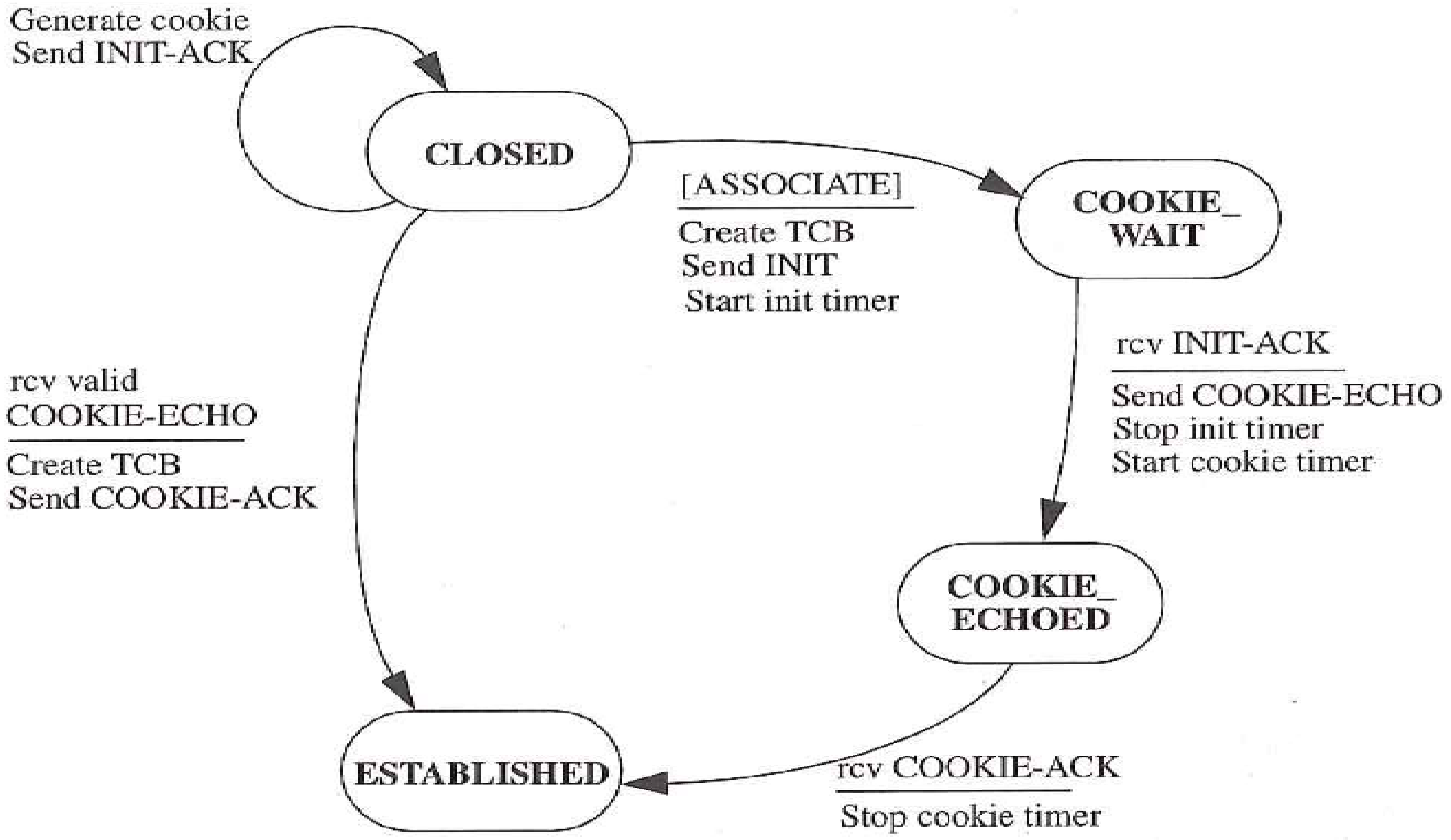


TCPDUMP – SCTP session

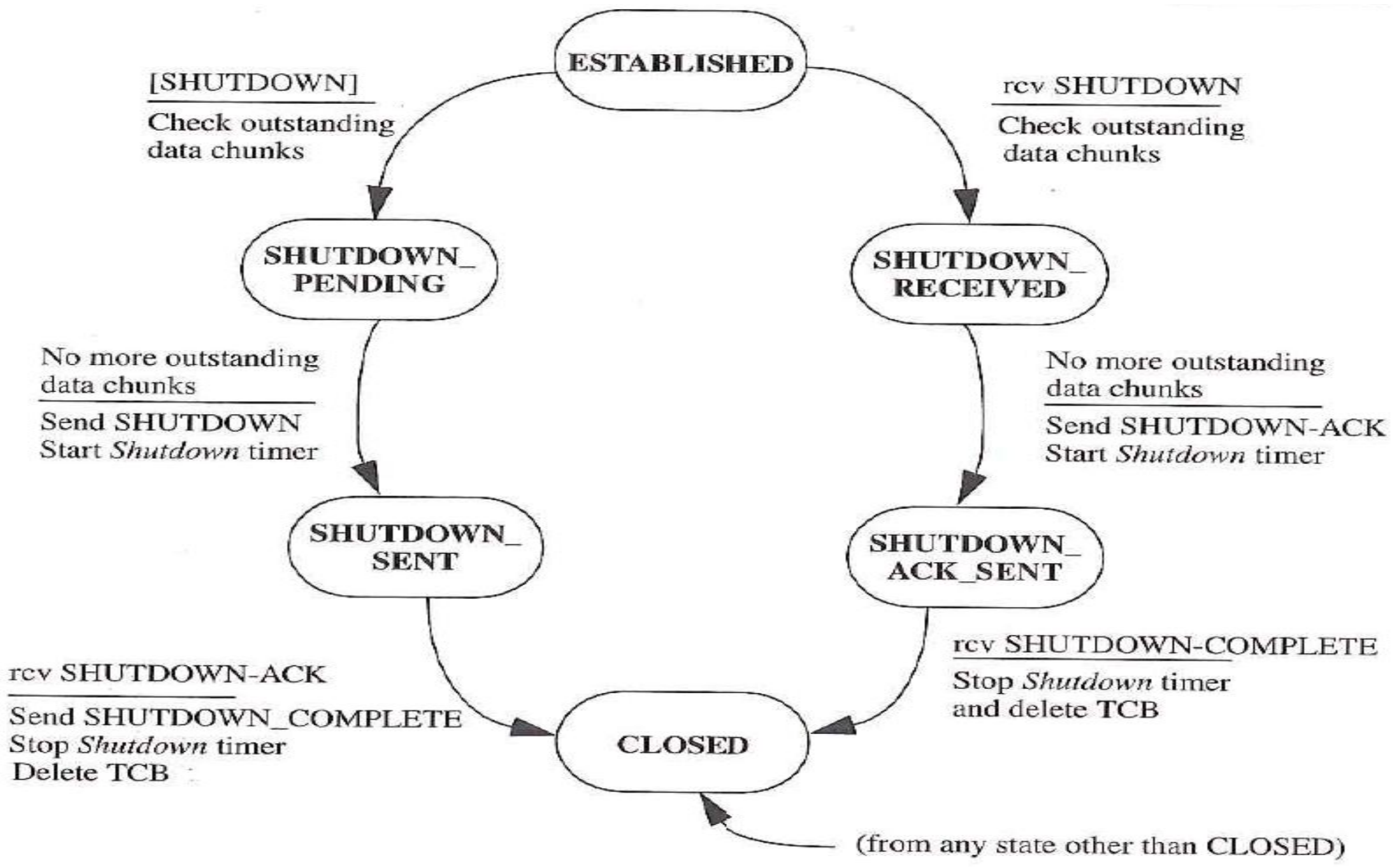
```
1.  tcpdump: verbose output suppressed, use -v or -vv for full
    protocol decode
2.  listening on lo, link-type EN10MB (Ethernet), capture size 65535
    bytes
3.  16:39:46.524327 IP6 ::1.38648 > ::1.13: sctp (1) [INIT] [init
    tag: 914475347] [rwnd: 106496] [OS: 10] [MIS: 65535] [init TSN:
    899852372]
4.  16:39:46.524477 IP6 ::1.13 > ::1.38648: sctp (1) [INIT ACK] [init
    tag: 657330372] [rwnd: 106496] [OS: 10] [MIS: 10] [init TSN:
    2650380151]
5.  16:39:46.524564 IP6 ::1.38648 > ::1.13: sctp (1) [COOKIE ECHO]
6.  16:39:46.524683 IP6 ::1.13 > ::1.38648: sctp (1) [COOKIE ACK]
7.  16:39:46.524879 IP6 ::1.13 > ::1.38648: sctp (1) [DATA] (B)(E)
    [TSN: 2650380151] [SID: 0] [SSEQ 0] [PPID 0x0]
8.  16:39:46.524929 IP6 ::1.38648 > ::1.13: sctp (1) [SACK] [cum ack
    2650380151] [a_rwnd 106470] [#gap acks 0] [#dup tsns 0]
9.  16:39:46.524990 IP6 ::1.13 > ::1.38648: sctp (1) [SHUTDOWN]
10. 16:39:46.525023 IP6 ::1.38648 > ::1.13: sctp (1) [SHUTDOWN ACK]
11. 16:39:46.525068 IP6 ::1.13 > ::1.38648: sctp (1) [SHUTDOWN
    COMPLETE]
```


CLOSED
Klient
Serwer
socket()
socket(), bind(), listen()
connect() – blokuje aktywne otwarcie
COOKIE-WAIT
INIT(Ta, J)
accept() – blokuje
CLOSED
Ta:INIT-ACK(Tz,K,cookie C)
Tz: COOKIE-ECHO C, DATA
accept() – powraca
read() - blokuje
ESTABLISHED
Ta: COOKIE-ACK, SACK, DATA
connect() –powraca aktywne otwarcie
ESTABLISHED
Tz: SACK, DATA
Ta: SACK
Tz: SHUTDOWN
passive close
read() returns 0
close
SHUTDOWN-ACK-SENT
Ta: SHUTDOWN-ACK
CLOSED
Tz: SHUTDOWN-COMPLETE
CLOSED

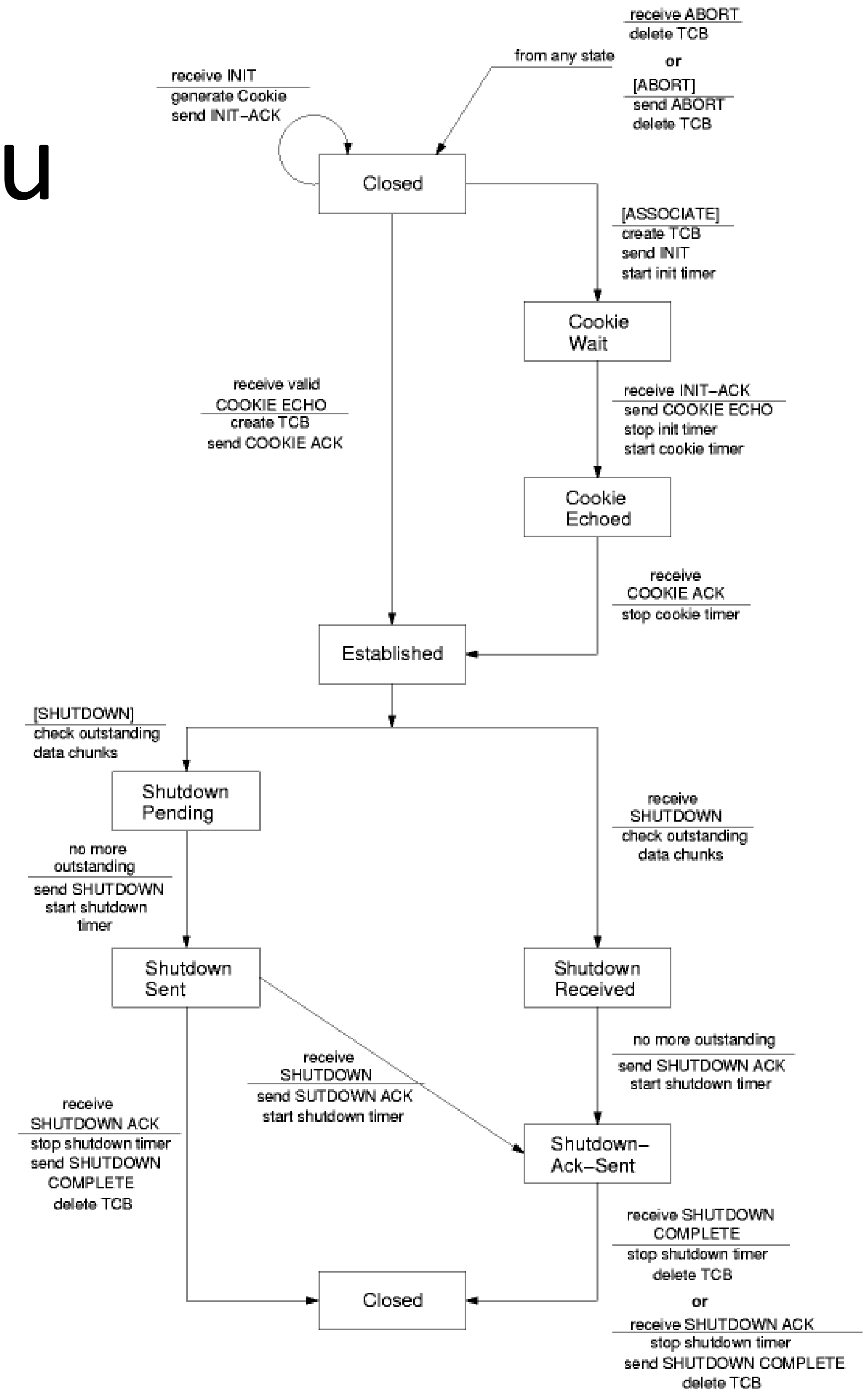
Stany protokołu SCTP: CLOSED -> ESTABLISHED



Stany protokołu SCTP: ESTABLISHED -> CLOSED



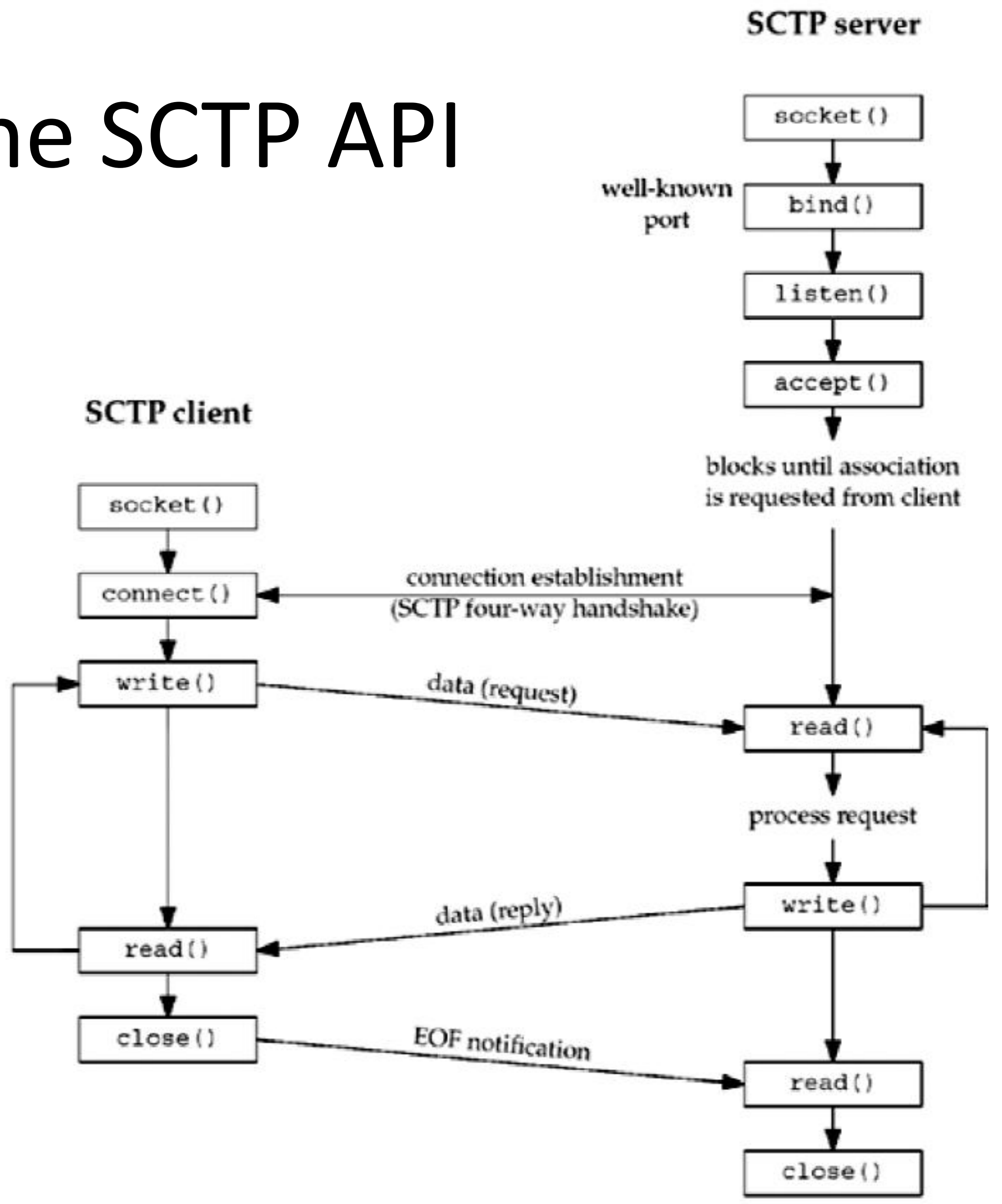
Stany protokołu SCTP



Modele obsługi

- *one-to-one socket* - „TCP-style” - dokładnie jedna asocjacja na gnieździe
- *one-to-many socket* - „ UDP-style” - kilka asocjacji może zostać ustanowionych jednego dla gniazda jednocześnie – tak jak dla UDP (dla gniazda niepołączonego) gniazdo może na przemian otrzymywać datagramy of różnych systemów końcowych

One-to-one SCTP API „TCP like”



One-to-one SCTP – różnice z TCP

1. Struktura aplikacji jest podobna ale gdy przerabiamy aplikację TCP na SCTP to wszystkie opcje gniazd muszą być zastąpione odpowiednikami dla protokołu SCTP o ile istnieją: np. TCP_NODELAY i TCP_MAXSEG na SCTP_NODELAY i SCTP_MAXSEG.
2. Strumień bajtów versus strumień dtagramów: SCTP zachowuje informacje o końcach komunikatów, tzn. że podczas jednego wywołania funkcji read() nie odbierzemy danych z dwóch komunikatów.
3. Gdy TCP używa zamknięcia jednego kierunku transmisji, to należy przerobić aplikację ponieważ SCTP tej funkcjonalności nie wspiera.
4. Mogą być używane funkcje do wysyłania z ustawionym adresem (np. sendto) ale wtedy nadpisujemy primary destination address.
5. Gniazdo SCTP one-to-one jest gniazdem (z rodziny AF_INET lub AF_INET6), o typie SOCK_STREAM i protokole IPPROTO_SCTP.

SCTP Example #1: One-to-one SCTP

- Kod programu:
  - TCP like (One-to-one):
    - Klient
    - Serwer
  - Działanie
  - Stany protokołu
  - Podgląd wymiany pakietów

One-to-many style

- The one-to-many style provides an application writer the ability to write a server without managing a large number of socket descriptors.
- A single socket descriptor will represent multiple associations, much the same way that a UDP socket can receive messages from multiple clients. An association identifier is used to identify a single association on a one-to-many style socket.
- A one-to-many-style SCTP socket is an IP socket (family AF_INET or AF_INET6) with type **SOCK_SEQPACKET** and protocol **IPPROTO_SCTP**

Users of the one-to-many style should keep the following issues in mind (1/3):

- When the client closes the association, the server side will automatically close as well, thus removing any state for the association inside the kernel.
- Using the one-to-many style is the only method that can be used to cause data to be piggybacked on the third or fourth packet of the four-way handshake
- Any **sendto, sendmsg, or sctp_sendmsg** to an address for which an association does not yet exist will cause an **active open** to be attempted, thus creating (if successful) a new association with that address. This behavior occurs even if the application doing the send has called the listen function to request a passive open.

Users of the one-to-many style should keep the following issues in mind (2/3):

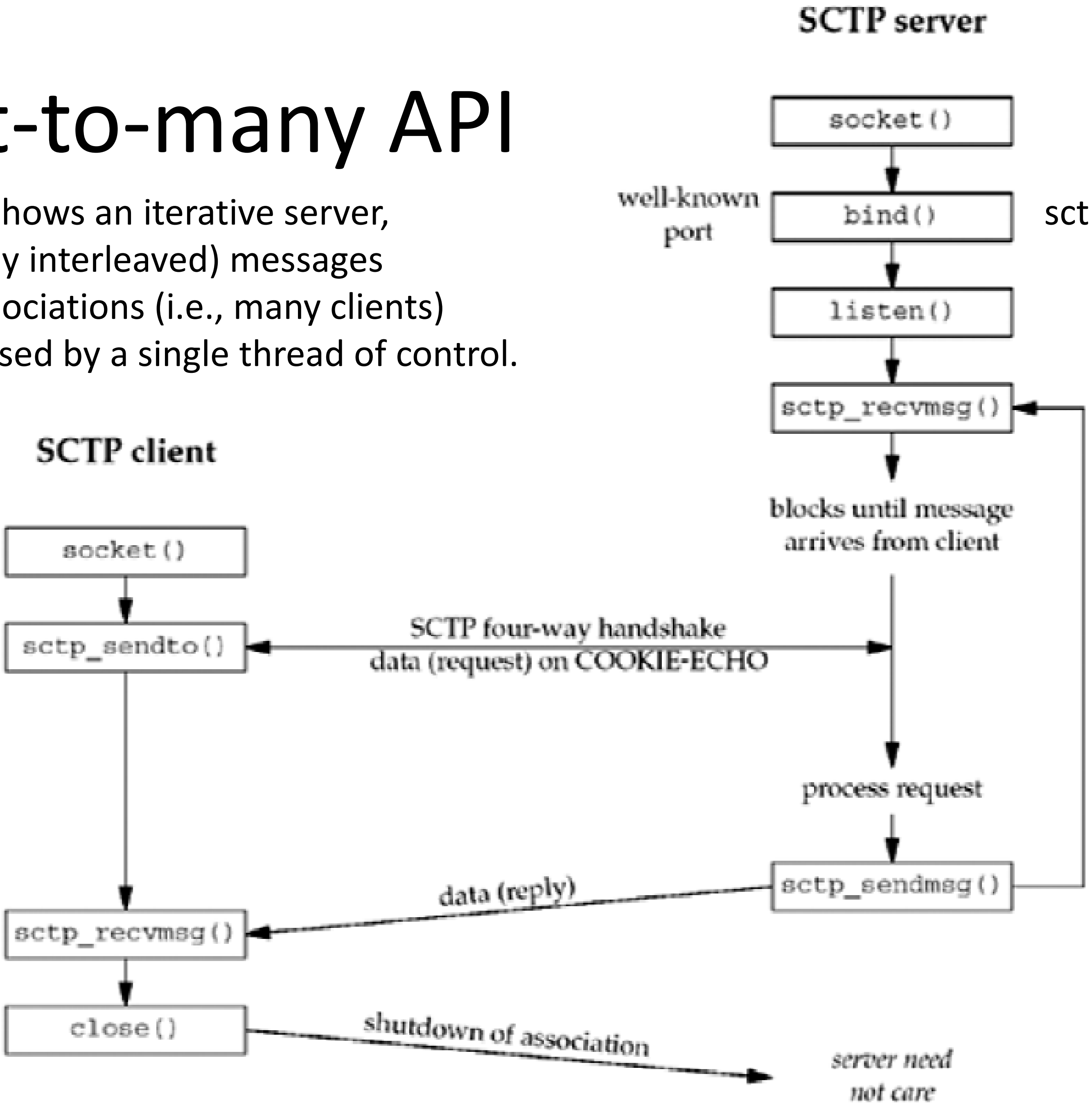
- The user must use the sendto, sendmsg, or sctp_sendmsg functions, and may not use the send or write function. (If the sctp_peeloff function is used to create a one-to-onestyle socket, send or write may be used on it.)
- Anytime one of the send functions is called, the primary destination address that was chosen by the system at association initiation time (Section 2.8) will be used unless the **MSG_ADDR_OVER** flag is set by the caller in a supplied sctp_sndrcvinfo structure. To supply this, the caller needs to use the sendmsg function with ancillary data, or the sctp_sendmsg function.

Users of the one-to-many style should keep the following issues in mind (3/3):

- Association events may be enabled, so if an application does not wish to receive these events, it should disable them explicitly using the **SCTP_EVENTS** socket option. By default, the only event that is enabled is the **sctp_data_io_event**, which provides ancillary data to the recvmsg and sctp_recvmsg call. This default setting applies to both the **one-to-one** and **one-to-many** style.

Onet-to-many API

This example shows an iterative server, where (possibly interleaved) messages from many associations (i.e., many clients) can be processed by a single thread of control.



sctp_bindx()

TCPDUMP – one-to-many SCTP session

```
1. 18:47:02.461025 IP6 fc00:1:1::24.57547 > fc00:1:1::23.7: sctp (1)
   [INIT] [init tag: 2404030505] [rwnd: 62464] [OS: 10] [MIS: 65535] [init
   TSN: 961390238]
2. 18:47:02.461359 IP6 fc00:1:1::23.7 > fc00:1:1::24.57547: sctp (1) [INIT
   ACK] [init tag: 3639668161] [rwnd: 62464] [OS: 10] [MIS: 10] [init TSN:
   3371930495]
3. 18:47:02.461982 IP6 fc00:1:1::24.57547 > fc00:1:1::23.7: sctp (1)
   [COOKIE ECHO] , (2) [DATA] (B)(E) [TSN: 961390238] [SID: 4] [SSEQ 0]
   [PPID 0x0]
4. 18:47:02.462341 IP6 fc00:1:1::23.7 > fc00:1:1::24.57547: sctp (1)
   [COOKIE ACK] , (2) [SACK] [cum ack 961390238] [a_rwnd 62455] [#gap acks
   0] [#dup tsns 0]
5. 18:47:02.462965 IP6 fc00:1:1::23.7 > fc00:1:1::24.57547: sctp (1)
   [DATA] (B)(E) [TSN: 3371930495] [SID: 5] [SSEQ 0] [PPID 0x0]
6. 18:47:02.463550 IP6 fc00:1:1::24.57547 > fc00:1:1::23.7: sctp (1)
   [SACK] [cum ack 3371930495] [a_rwnd 62455] [#gap acks 0] [#dup tsns 0]
7. 18:47:04.404011 IP6 fc00:1:1::24.57547 > fc00:1:1::23.7: sctp (1)
   [SHUTDOWN]
8. 18:47:04.404124 IP6 fc00:1:1::23.7 > fc00:1:1::24.57547: sctp (1)
   [SHUTDOWN ACK]
9. 18:47:04.404576 IP6 fc00:1:1::24.57547 > fc00:1:1::23.7: sctp (1)
   [SHUTDOWN COMPLETE]
```

SCTP API

- **sctp_bindx()**
- **sctp_connectx()**
- **sctp_recvmsg()**
- **sctp_sendmsg()**
- **sctp_send()**
- **sctp_opt_info()**
- **sctp_peeloff()**
- **sctp_getpaddrs() / sctp_freepaddrs()**
- **sctp_getladdrs() / sctp_freeladdrs()**

SCTP API: sctp_bindx()

int sctp_bindx(int sd, struct sockaddr * addrs, int addrcnt, int flags);

- Binds to several addreses
- If application binds to single address or to all addresses it can use bind() function
- For IPv4 socket sockaddr_in address structure have to be uses, **for IPv6 both sockaddr_in or sockaddr_in6** are possible
- The *flags* parameter is formed from performing the bitwise OR operation on zero or more of the following currently defined flags:
  - SCTP_BINDX_ADD_ADDR
  - SCTP_BINDX_REM_ADDR

SCTP API: sctp_connectx()

int sctp_connectx(int sd, struct sockaddr * addrs, int addrcnt, sctp_assoc_t * id);

- addrs - pointer to an array of one or more socket addresses
- addrcnt – number of addresses in addrs array
- id – association id returned by the kernel

SCTP API: sctp_recvmsg()

ssize_t sctp_recvmsg(int *s*, void **msg*, size_t *len*, struct sockaddr **from*, socklen_t **fromlen*, struct sctp_sndrcvinfo **sinfo*, int**msg_flags*);

- The **sctp_recvmsg()** function enables receipt of a message from the SCTP endpoint specified by the *s* parameter. The calling program can specify the following attributes:
- *msg* - the address of the message buffer.
- *len* - the length of the message buffer.
- *from* - the pointer to an address that contains the sender's address.
- *fromlen* - this parameter is the size of the buffer associated with the address in the *from* parameter.
- *sinfo* - only active if the calling program enables sctp_data_io_events. To enable sctp_data_io_events, call the **setsockopt()** function with the socket option SCTP_EVENTS. When sctp_data_io_events is enabled, the application receives the contents of the sctp_sndrcvinfo structure for each incoming message.

sctp_data_io_event (netinet/sctp.h)

```
struct sctp_sndrcvinfo {
        __u16 sinfo_stream;
        __u16 sinfo_ssn;
        __u16 sinfo_flags;
        __u32 sinfo_ppid;
        __u32 sinfo_context;
        __u32 sinfo_timetolive;
        __u32 sinfo_tsn;
        __u32 sinfo_cumtsn;
        sctp_assoc_t  sinfo_assoc_id;
};
```

SCTP API: sctp_sendmsg() (1/3)

ssize_t sctp_sendmsg(int *s*, const void **msg*, size_t *len*, const struct sockaddr **to*, socklen_t *to_len*, uint32_t *ppid*, uint32_t*flags*, uint16_t *stream_no*, uint32_t *timetolive*, uint32_t *context*);

- *s* - the SCTP socket
- *msg* - the message sent by the **sctp_sendmsg()** function.
- *len* - the length of the message. This value is expressed in bytes.
- *to* - the destination address of the message.

SCTP API: sctp_sendmsg() (2/3)

- *to_len* - the length of the destination address.
- *ppid* - the application-specified payload protocol identifier.
- *stream_no* - the target stream for this message.
- *timetolive* - the time period after which the message expires if it has not been successfully sent to the peer. This value is expressed in milliseconds.
- *context* – the value returned if an error occurs during the sending of the message.

SCTP API: sctp_sendmsg() (3/3)

flags :

- **MSG_UNORDERED** - When is set, the function delivers the messages unordered.
- **MSG_ADDR_OVER** - When is set, the **sctp_sendmsg()** function uses the address in the *to* parameter instead of the association's primary destination address. Only used with one-to-many style SCTP sockets.
- **MSG_ABORT** - When is set, the specified association aborts by sending an ABORT signal to its peer. Only used with one-to-many style SCTP sockets.
- **MSG_EOF** - When is set, the specified association enters graceful shutdown. Only with one-to-many style SCTP sockets.
- **MSG_PR_SCTP** - when is set, the message expires when its transmission has not successfully completed within the time period specified in the timetolive parameter.

SCTP API: sctp_send ()

ssize_t sctp_send(int *s*, const void **msg*, size_t *len*, const struct sctp_sndrcvinfo **sinfo*, int *flags*);

The **sctp_send()** function is usable by one-to-one and one-to-many style sockets. The **sctp_send()** function enables advanced SCTP features while sending a message from an SCTP endpoint.

- *s* - specifies the socket created by the **socket()** function.
- *msg* - the message sent by the **sctp_send()** function.
- *len* - the length of the message. This value is expressed in bytes.
- *sinfo* - the parameters used to send the message. For a one-to-many style socket, this value can contain the association ID to which the message is being sent.
- *flags* - identical to the flags parameter in the **sendmsg()** function.

SCTP API: sctp_peeloff()

int **sctp_peeloff**(int *sock*, sctp_assoc_t *id*);

sock - one-to-many style socket descriptor

id - the identifier of the association to branch off to a separate file descriptor

- With SCTP, a one-to-many socket can also be used in conjunction with the sctp_peeloff function to allow the iterative and concurrent server models to be combined as follows:
- The sctp_peeloff function can be used to peel off a particular association (for example, a long-running session) from a one-to-many socket into its own one-to-one socket.
- The one-to-one socket of the extracted association can then be dispatched to its own thread or forked process (as in the concurrent model).
- Meanwhile, the main thread continues to handle messages from any remaining associations in an iterative fashion on the original socket.
- A one-to-many-style SCTP socket is an IP socket (family AF_INET or AF_INET6) with type SOCK_SEQPACKET and protocol IPPROTO_SCTP

SCTP API: sctp_getpaddr()

- The **sctp_getpaddrs()** function returns all peer addresses in an association.

int **sctp_getpaddrs**(int *sock*, sctp_assoc_t *id*, void ***addrs*);

- The **sctp_freepaddrs()** function frees all of the resources that were allocated by a previous call to the **sctp_getpaddrs()**.

void sctp_freepaddrs(void **addrs*);

SCTP API: sctp_getladdr()

- The **sctp_getladdrs()** function returns all locally bound addresses on a socket:

*int sctp_getladdrs(int sock, sctp_assoc_t id, void **addrs);*

- The **sctp_freeladdrs()** function frees all of the resources that were allocated by a previous call to the **sctp_getladdrs()**.

*void sctp_freeladdrs(void *addrs);*

SCTP Example #2

1. SCTP – one-to-many
2. peel_off() i fork()

sctpclientv4_01.c

sctpservv4_01.c

sctpservv4_fork_02.c

Opcje protokołu i asocjacji SCTP

- getsockopt()
- setsockopt()
- sctp_opt_info() – odnośnie asocjacji

  int sctp_opt_info(int *sock*, sctp_assoc_id_t *id*, int *opt*, void **arg*, socklen_t **len*);
- Notyfikacje związane z asocjacją – otrzymywane w funkcjach odbierających
- Plik nagłówkowy: /usr/include/netinet/**sctp.h**

SCTP options

- SCTP_ASSOCINFO - Returns the association-specific parameters.
- SCTP_DEFAULT_SEND_PARAM - Returns the default set of parameters that a call to the sendto() function uses on this association
- SCTP_PEER_ADDR_PARAMS - Returns the parameters for a specified peer address.
- SCTP_STATUS – returns the current status information about the association (e.g. states)
- SCTP_NODELAY - Turn on/off any Nagle-like algorithm.
- SCTP_AUTOCLOSE - one-to-many style socket only. Cause associations that are idle for more than the specified number of seconds to automatically close.
- SCTP_SET_PEER_PRIMARY_ADDR - Requests that the peer mark the enclosed address as the association primary.
- SCTP_PRIMARY_ADDR - Requests that the local SCTP stack use the enclosed peer address as the association primary.

SCTP options

- SCTP_RTOINFO - Set/get the protocol parameters that are used to initialize and bind the retransmission timeout (RTO) tunable.
- SCTP_INITMSG - This option is used to get or set the protocol parameters for the default association initialization.
- **SCTP_EVENTS** - is used to specify various notifications and ancillary data the user wishes to receive.
- SCTP_MAXSEG - specifies the maximum size to put in   any outgoing SCTP DATA chunk. If a message is larger than this size it will be fragmented by SCTP into the specified size.
- SCTP_DISABLE_FRAGMENTS - If enabled no SCTP message fragmentation will be performed.
- SCTP_GET_ASSOC_STATS Applications can retrieve current statistics about an association, including SACKs sent and received, SCTP packets sent and received.

SCTP API: sctp_opt_info()

int sctp_opt_info(int *sock*, sctp_assoc_id_t *id*, int *opt*,
void **arg*, socklen_t **len*);

- The **sctp_opt_info()** function returns the SCTP level options that are associated with the socket described in the *sock* parameter. If the socket is a one-to-many style SCTP socket the value of the ***id*** parameter refers to a specific association. The ***id*** parameter is ignored for one-to-one style SCTP sockets. The value of the ***opt*** parameter specifies the SCTP socket option to get. The value of the ***arg*** parameter is an option-specific structure buffer that is allocated by the calling program. The value of the ****len*** parameter is the length of the option.

SCTP Notifications

- Default: only **sctp_data_io_event**
- Required **SCTP_EVENTS** socket option set up
- Ten events
- **recvmsg()** function or the **sctp_recvmsg()** function. When the data returned is an event notification, the msg_flags field of these two functions will contain the **MSG_NOTIFICATION** flag
- Each type of notification is in **tag-length-value** (**TLV**) form

SCTP notification event

```
struct sctp_event_subscribe {
    __u8 sctp_data_io_event;
    __u8 sctp_association_event;
    __u8 sctp_address_event;
    __u8 sctp_send_failure_event;
    __u8 sctp_peer_error_event;
    __u8 sctp_shutdown_event;
    __u8 sctp_partial_delivery_event;
    __u8 sctp_adaptation_layer_event;
    __u8 sctp_authentication_event;
    __u8 sctp_sender_dry_event;
};
```

sctp_data_io_event (netinet/sctp.h)

```
struct sctp_sndrcvinfo {
      __u16 sinfo_stream;
      __u16 sinfo_ssn;
      __u16 sinfo_flags;
      __u32 sinfo_ppid;
      __u32 sinfo_context;
      __u32 sinfo_timetolive;
      __u32 sinfo_tsn;
      __u32 sinfo_cumtsn;
      sctp_assoc_t   sinfo_assoc_id;
};
```

SCTP summary

- SCTP provides the application writer with two different interface styles:
  - the **one-to-one** style, mostly compatible with existing TCP applications to ease migration to SCTP,
  - the **one-to many** style, allowing access to all of SCTP's features.
- The **sctp_peeloff** function provides a method of extracting an association from one style to the other. SCTP also provides numerous notifications of transport events to which an application may wish to subscribe. These events can aid an application in better managing the associations it maintains.
- Since SCTP is multihomed, not all the standard sockets functions are adequate. Functions like **sctp_bindx, sctp_connectx, sctp_getladdrs, and sctp_getpaddrs** provide methods to better control and examine the multiple addresses that can make up an SCTP association.
- Utility functions such as **sctp_sendmsg** and **sctp_recvmsg** can simplify the use of these advanced features.

Stream Control Transmission Protocol (SCTP) Specification

- http://www.iana.org/assignments/sctp-parameters/sctp-parameters.xhtml

SCTP Example #3

1. Powiadomienia (notifications)
2. Zmiana adresów w węzłach końcowych

sctpclientv6_01.c

sctpclientv6_02.c