SCTP Sockets

Introduction

- SCTP provides some features of both TCP and UDP.
- It is message-oriented, and provides a reliable sequenced delivery of messages.

- SCTP supports multi-homing, i.e., multiple IPs on both sides of the connection. So it is called an association instead of a connection, as a connection involves communication between two IPs, while an association refers to communication between two systems that may have multiple IPs.

- SCTP can provide multiple streams between connection endpoints, and each stream will have its own reliable sequenced delivery of messages, so any lost message will not block the delivery of messages in any of the other streams.

Interface Models

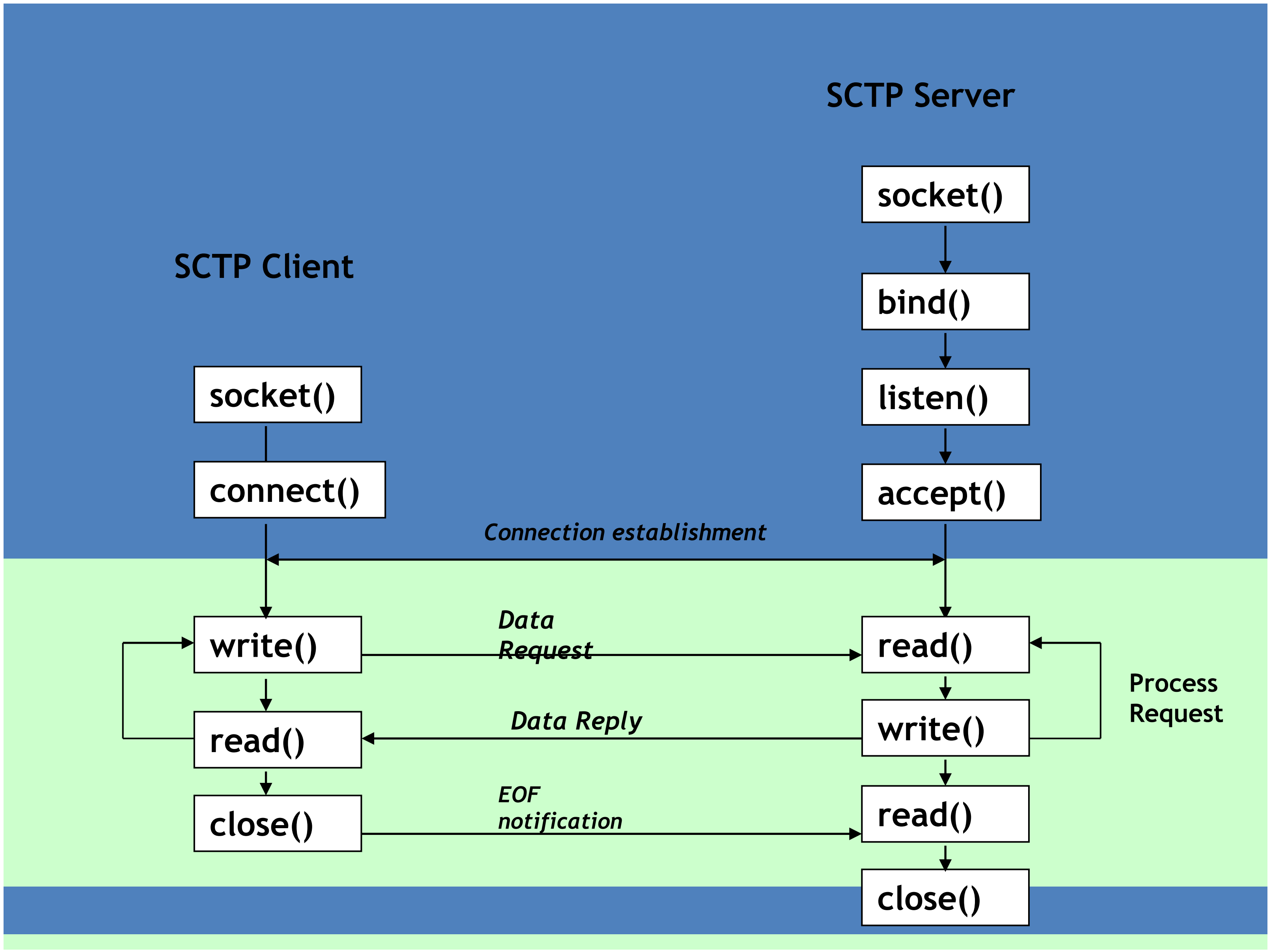
- SCTP sockets have two type of models:
    - one-to-one socket Model
    - one-to-many socket Model

- A one-to-one socket corresponds to exactly one SCTP association.
- The association identifier is a value of type sctp assoc.
- The client closes the association, then the server side will be automatically close.
- one-to-one socket that corresponds to exactly one SCTP association (similar to TCP).

- One-to-one sockets (also called TCP-style sockets) were developed to ease porting existing TCP applications to SCTP, so the difference between a server implemented using TCP and SCTP is not much.

- We just need to modify the call to socket() to socket(AF_INET, SOCK_STREAM, and IPPROTO_SCTP) while everything else stays the same — the calls to listen(), accept() for the server, and connect() for the client, with read() and write() calls for both.

Figure : Socket functions for SCTP one-to-one style.



Interface models

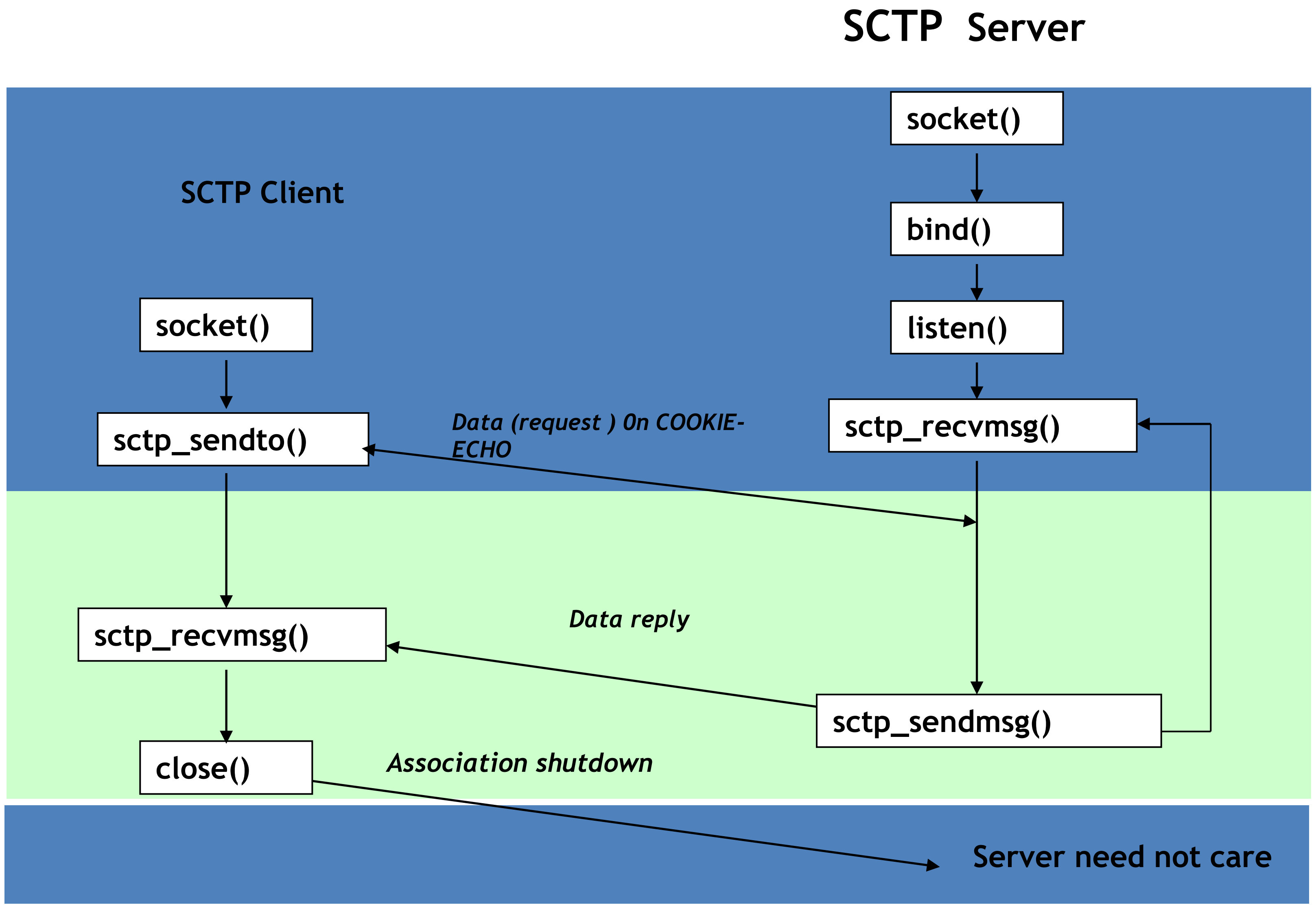
- A one-to-many socket, where many SCTP associations can be active on a socket simultaneously (similar to UDP receiving datagrams from several endpoints).

- The socket function call to specify IPPROTO_SCTP instead of IPPROTO_TCP as the third argument. Simply making this change, however, would not take advantage of any of the additional features provided by SCTP except multihoming.

- we use the sctp_sendmsg() and sctp_recvmsg() functions instead of read() and write().

- Using one-to-many style allows us to exercise all of SCTP's features

Figure: Socket functions for SCTP one-to-Many style.



let's jump to the **one-to-many** or UDP-style socket, and write a server using multiple streams that the client follows.

Sctpserver.c

```
#include <stdio.h>
#include <string.h>
#include <time.h>
#include <sys/socket.h>
#include <sys/types.h>
#include <netinet/in.h>
#include <netinet/sctp.h>
#include <arpa/inet.h>


#define MAX_BUFFER  1024

int main()
{
    int sfd, cfd, len, i;
    struct sockaddr_in saddr, caddr;
    struct sctp_initmsg initmsg;
    char buff[INET_ADDRSTRLEN];
    char buffer[MAX_BUFFER+1] = "Message
##\n";


    sfd = socket( AF_INET, SOCK_STREAM,
IPPROTO_SCTP );
```

```
    bzero( (void *)&saddr, sizeof(saddr) );
    saddr.sin_family = AF_INET;
    saddr.sin_addr.s_addr = htonl(
INADDR_ANY );
    saddr.sin_port = htons(29008);

    bind( sfd, (struct sockaddr *)&saddr,
sizeof(saddr) );

/* Maximum of 3 streams will be available
per socket */
    memset( &initmsg, 0, sizeof(initmsg) );
    initmsg.sinit_num_ostreams = 3;
    initmsg.sinit_max_instreams = 3;
    initmsg.sinit_max_attempts = 2;
    setsockopt( sfd, IPPROTO_SCTP,
SCTP_INITMSG,
            &initmsg, sizeof(initmsg) );

    listen( sfd, 5 );
```

Sctpserver.c contd…..

```
for(;;) {
      printf("Server Running\n");

      len=sizeof(caddr);
      cfd=accept(sfd, (struct sockaddr *)&caddr, &len);

      printf("Connected to %s\n",
         inet_ntop(AF_INET, &caddr.sin_addr, buff,
         sizeof(buff)));

      for(i=0; i< 3; i++) {
/* Changing 9th character the character after # in the message buffer */
         buffer[9] = '1'+i;

         sctp_sendmsg( cfd, (void *)buffer, (size_t)strlen(buffer),
               NULL, 0, 0, 0, i /* stream */, 0, 0 );
         printf("Sent: %s\n", buffer);
      }

      close( cfd );
   }
   return 0;
}
```

Sctpclient.c

```
#include <stdio.h>
#include <string.h>
#include <sys/socket.h>
#include <sys/types.h>
#include <netinet/in.h>
#include <netinet/sctp.h>

#define MAX_BUFFER  1024

int main(int argc, char **argv)
{
    int cfd, i, flags;
    struct sockaddr_in saddr;
    struct sctp_sndrcvinfo sndrcvinfo;
    struct sctp_event_subscribe events;
    struct sctp_initmsg initmsg;
    char buffer[MAX_BUFFER+1];

    if(argc!=2) {
        printf("Usage: %s ipaddress\n",
argv[0]);
    return -1;
    }
     cfd = socket( AF_INET,
SOCK_STREAM, IPPROTO_SCTP );
```

```
/* Specify that a maximum of 3 streams will
be available per socket */
    memset( &initmsg, 0, sizeof(initmsg) );
    initmsg.sinit_num_ostreams = 3;
    initmsg.sinit_max_instreams = 3;
    initmsg.sinit_max_attempts = 2;
    setsockopt( cfd, IPPROTO_SCTP,
SCTP_INITMSG,
            &initmsg, sizeof(initmsg) );

    bzero( (void *)&saddr, sizeof(saddr) );
    saddr.sin_family = AF_INET;
    inet_pton(AF_INET, argv[1],
&saddr.sin_addr);
    saddr.sin_port = htons(29008);

    connect( cfd, (struct sockaddr *)&saddr,
sizeof(saddr) );

 memset( (void *)&events, 0, sizeof(events) );
    events.sctp_data_io_event = 1;
    setsockopt( cfd, SOL_SCTP, SCTP_EVENTS,
        (const void *)&events, sizeof(events) );
```

Sctpclient.c contd….

```
/* Sending three messages on different streams */

   for (i=0; i<3; i++) {
      bzero( (void *)&buffer, sizeof(buffer) );

      sctp_recvmsg( cfd, (void *)buffer, sizeof(buffer),
                  (struct sockaddr *)NULL, 0, &sndrcvinfo,
&flags );

      printf("Received following data on stream %d\n\n%s\n",
         sndrcvinfo.sinfo_stream, buffer);

   }

   close(cfd);

   return 0;
}
```

- The server is sending three messages on three different streams, and the client is just receiving the messages and printing them on the screen

- The code is similar to the TCP client, as we are again making calls to the same functions.

- The difference is that we are creating an iterative server, similar to the one for UDP, but we have an **accept()** call here.

- Lets try to understand the functions used in program

Socket creation on both Server and Client side using following syntax

```
sfd = socket(AF_INET, SOCK_STREAM,
IPPROTO_SCTP)
```

```
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/sctp.h>

int sctp_sendmsg (int sd, const void *msg, size_t len,
    struct sockaddr *to, socklen_t tolen, uint32_t ppid, uint32_t
    flags, uint16_t stream_no, uint32_t timetolive, uint32_t context);
```

- We are using this function to send a message from a socket while using the advanced features of SCTP. The first argument to the function is sd, the socket descriptor, from which the message msg of length len is sent.

- The fourth argument is to give the destination address — tolen specifies the length of the destination address, while stream_no identifies the stream number to send this message to. The flags parameter is used to send some options to the receiver. You can check out the manual pages for sctp_sendmsg().

- The timetolive parameter is time in milliseconds after which the message will expire if not sent by then; the zero here indicates that no time-out is set. The context is the value passed to the upper layer along with the undelivered message, if an error occurs while sending the message. When successful, it will return the number of bytes sent, or -1 on error.

```
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/sctp.h>

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

- This function does the reverse of the **sctp_sendmsg** function and is used to receive a message. The parameters are similar.

- The socket **sd** receives the **msg** of length **len** from the address ***from** with a length ***fromlen**, and ***sinfo** is a pointer to the address, which will be filled upon receipt of the message. **mag_flags** is a pointer to an integer with flags like **MSG_NOTIFICATION** or **MSG_EOR**.

- It returns the number of bytes received, or -1 on error.

```
#include <sys/types.h>
#include <sys/socket.h>

int setsockopt(int sockfd, int level, int optname,
                      const void *optval, socklen_t optlen);
```

- This function is used to set the options for the socket sockfd. The next argument is the level at which the option resides.
- To manipulate options at the sockets API level, the level is specified as SOL_SOCKET. optname and any specified options are passed uninterpreted to the appropriate protocol module for interpretation.
- The arguments optval and optlen are used to access option values for setsockopt() that are stored in the structure. The options we set in the server are:
  - initmsg.sinit_num_ostreams = 3;
  - initmsg.sinit_max_instreams = 3;
  - initmsg.sinit_max_attempts = 2;
- Here, the first two lines tell us that the output and input streams available are three, and the maximum attempts will be two. The same options are set in the client program. Other options are set for the events.
- This structure will be filled when an event like "message received" occurs, and our program is notified.
- Its counterpart function is getsockopts().

Thank You