



SCTP: An Overview

Part 3: Socket API

Randall Stewart, Cisco Systems

Phill Conrad, University of Delaware

Outline

10h00-11h00	intro	Randy
	overview of SCTP What is SCTP? What are the major features?	Phill
11h15-12h15	SCTP details	Randy
13h15-14h15	details of sockets API (Randy)	Phill or Randy
	open Q and A	Both



Sockets API

- **Chapter 11 of the SCTP book discusses the socket API. This text is quite dated, but gives the reader a general idea how the socket API works.**
- **However, a better reference for SCTP socket API programming is the third revision of Stevens' Unix Network Programming.**
- **This new book has three comprehensive up-to-date chapters that detail the finer points of working with the SCTP socket API.**

SCTP Socket Types

- SCTP socket API comes in two forms: **one-to-one** and **one-to-many**.
- The **one-to-many** at one time was known by the “UDP style” socket. The **one-to-one** used to be called the a “TCP style” socket.
- So what is the purpose of each socket style and how can it be used?
- We will start with the **one-to-one** style.

One-to-One style

- **The purpose of the one-to-one style socket is to provide a smooth transition mechanism for those applications running on TCP and wishing to move to SCTP.**
- **The same semantics used in TCP are used with this style.**
- **A server will typically open the socket, make a call to listen (to accept associations), and call accept, blocking upon the arrival of a new association.**

One-to-One style

- **The only notable difference between a TCP socket and a SCTP socket is the socket call uses IPPROTO_SCTP instead of IPPROTO_TCP (or 0).**
- **Two other common socket options that might be used in a TCP application have SCTP equivalents:**
 - TCP_NODELAY -> SCTP_NODELAY**
 - TCP_MAXSEG -> SCTP_MAXSEG**
- **SCTP has a host of other socket options as well which we will touch on further on.**

One-to-One Style

- **Switching from TCP to SCTP becomes easy with this style of socket due to the few number of changes that have to be made.**
- **To give you an idea on this, note that I ported a version of mozilla with only two lines of change.**
- **Of course a quick change like I did in mozilla did not gain useage of SCTP streams but it does gain you the multi-homing aspects of SCTP.**
- **Other cavets of moving a TCP application are that there is NO half-close state, so if an application makes use of this, that code will need to be re-written.**

One-to-One Style

- **Another thing that MAY be an issue is that some TCP applications will write a 2 or 4 byte record length followed by that many bytes of data. If an application behaves in this way SCTP will make each write a single message.**
- **These two message would most likely be bundled together but overall this increases the overhead on the wire.**
- **So what does a typical application using the one-to-one style socket look like?**

One-to-One Example Server

```
int sd, newfd, sosz;
struct sockaddr_in6 sin6;
sosz = sizeof(sin6);
sd = socket(AF_INET6, SOCK_STREAM, IPPROTO_SCTP);
listen(sd, 1);
while (1) {
    newfd = accept(sd, (struct sockaddr *)&sin6, &sosz)
    do_child_stuff(newfd, &sin6, sosz);
}
```

One-to-Many style

- **A typical server using a one-to-many style socket will do a `socket()` call, followed by a `listen()` and `recvfrom()`.**
- **A typical client will just `sendto()` the server of his choice.**
- **Note that the `connect()` and `accept()` call are not needed.**
- **The `connect()` call can be done by either side (server or client) but it is not needed.**

One-to-Many style

- **Note that this style is more like what a UDP client/server would look like thus the previous name.**
- **So what does a typical one-to-many style server look like?**

One-to-many Example Server

```
int sd, newfd, sosz, msg_flags;
struct sockaddr_in6 sin6;
struct sndrcvinfo snd_rcv;
char buf[8000];
sosz = sizeof(sin6);
sd = socket(AF_INET6, SOCK_SEQPKT, IPPROTO_SCTP);
listen(sd, 1);
while (1) {
    len = sctp_recvmsg(sd, buf, sizeof(buf), (sockaddr *)&sin6,
        &sosz,
        &snd_rcv, &msg_flags);
    do_child_stuff(newfd, buf, len, &sin6, &snd_rcv, msg_flags);
}
```

One-to-many Description

- Note in the previous example we introduced the first of several new/extra calls **sctp_recvmsg()**.
- This call is usually built as a library call (its not a true system call in most cases).
- It provides a convenience function that makes it easy to find out specific information about stream id and other auxiliary information that SCTP can provide upon receiving messages.
- But before we get in to the details of all the extra calls we need to discuss notifications.

SCTP Notifications

- **The SCTP stack, at times, has information it may wish to share with its application (or Upper Layer Protocol ... ULP).**
- **The ULP can turn off and on specific notifications via a socket options call.**
- **By default ALL notifications are off.**
- **So how does one get a notification?**
- **By reading data and looking at the msg_flags, if the message read is a notification, then "MSG_NOTIFICATION" is contained within the msg_flags argument upon return.**

More on Notifications

- **If the user does NOT use the `sctp_rcvmsg()` call, then you can also gain access to this flag using the `rcvmsg()` system call and look at the `msg.msg_flags` field (most library calls implementing `sctp_rcvmsg()` use `rcvmsg()` and copy the `msg.msg_flags` into the `int*` passed to `sctp_rcvmsg()`).**
- **So what do you get when you read a notification?**
- **A union is read in that looks as follows:**

Notification Union

```
/* notification event */
union sctp_notification {
    struct sctp_tlv sn_header;
    struct sctp_assoc_change sn_assoc_change;
    struct sctp_paddr_change sn_paddr_change;
    struct sctp_remote_error sn_remote_error;
    struct sctp_send_failed      sn_send_failed;
    struct sctp_shutdown_event
sn_shutdown_event;
    struct sctp_adaption_event sn_adaption_event;
    struct sctp_pdapi_event sn_pdapi_event;
};
```

Deciphering Notifications

- **Every Notification uses a TLV format as illustrated below:**

```
struct sctp_tlv {  
    u_int16_t sn_type;  
    u_int16_t sn_flags;  
    u_int32_t sn_length;  
};
```

- **So what type of notifications do you get?**

Association change

- **SCTP_ASSOC_CHANGE** - indicates that a change has occurred in regard to an association (e.g. a new association is now present on the socket or an association has went away/failed).

```
struct sctp_assoc_change {
    u_int16_t sac_type;
    u_int16_t sac_flags;
    u_int32_t sac_length;
    u_int16_t sac_state;
    u_int16_t sac_error;
    u_int16_t
sac_outbound_streams;
    u_int16_t sac_inbound_streams;
    sctp_assoc_t sac_assoc_id;
};
```

A Peer Address Change event

- An **SCTP_PEER_ADDR_CHANGE** will indicate that something has occurred with the address (in-service, out-of-service, added, deleted etc).

```
/* Address events */
struct sctp_paddr_change {
    u_int16_t spc_type;
    u_int16_t spc_flags;
    u_int32_t spc_length;
    struct sockaddr_storage
    spc_aaddr;
    u_int32_t spc_state;
    u_int32_t spc_error;
    sctp_assoc_t spc_assoc_id;
};
```

A Remote Error Event

- An **SCTP_REMOTE_ERROR** will communicate a remote error sent by the peer, this will be in the form of a TLV and may indicate some internal stack debugging information as to why an association was closed.

```
/* remote error events */
struct sctp_remote_error {
    u_int16_t sre_type;
    u_int16_t sre_flags;
    u_int32_t sre_length;
    u_int16_t sre_error;
    sctp_assoc_t
sre_assoc_id;
    u_int8_t sre_data[4];
};
```

Send Failure

- An **SCTP_SEND_FAILED** will indicate that data queued was not acknowledged by the peer and will include the actual data that was attempted to be sent (within some limits). This may occur due to partial reliability or right before an association comes down.

```
/* data send failure event */
struct sctp_send_failed {
    u_int16_t ssf_type;
    u_int16_t ssf_flags;
    u_int32_t ssf_length;
    u_int32_t ssf_error;
    struct sctp_sndrcvinfo ssf_info;
    sctp_assoc_t ssf_assoc_id;
    u_int8_t ssf_data[4];
};
```

Shutdown Event

- An **SCTP_SHUTDOWN_EVENT** indicates that a graceful shutdown as occurred on an association.

```
/* shutdown event */
struct sctp_shutdown_event {
    u_int16_t    sse_type;
    u_int16_t    sse_flags;
    u_int32_t    sse_length;
    sctp_assoc_t
    sse_assoc_id;
};
```

Adaption Layer Event

- An **SCTP_ADAPTION_INDICATION** is a part of the add-ip extension and allows an upper layer to communicate an integer at startup informing the peer what type of ULP is being operated (iSCSI, RDMA, ?)

```
/* Adaption layer indication stuff */
struct sctp_adaption_event {
    u_int16_t    sai_type;
    u_int16_t    sai_flags;
    u_int32_t    sai_length;
    u_int32_t    sai_adaption_ind;
    sctp_assoc_t sai_assoc_id;
};
```

Partial Delivery Event

- An **SCTP_PARTIAL_DELIVERY_EVENT** will indicate when something has went wrong on a partial delivery that has been begun (e.g. The association closed or the message was skipped via partial reliability).

```
/* pdapi indications */
struct sctp_pdapi_event {
    u_int16_t    pdapi_type;
    u_int16_t    pdapi_flags;
    u_int32_t    pdapi_length;
    u_int32_t    pdapi_indication;
    sctp_assoc_t pdapi_assoc_id;
};
```

Common to events is the `assoc_id`

- **Note that all events include something called an `assoc_id`.**
- **This is a unique identifier to the association.**
- **Many of the extended SCTP calls can use this for sending and or configuring an association with socket options.**
- **An application that wishes to use `assoc_id`'s needs to be aware of association id re-use and must pay close attention to failure and closing events.**

So how does one get notifications?

- **The socket option `SCTP_EVENTS` is used to turn on/off all of the various events by passing it the following structure:**

```
/* On/Off setup for subscription to events */
struct sctp_event_subscribe {
    u_int8_t sctp_data_io_event;
    u_int8_t sctp_association_event;
    u_int8_t sctp_address_event;
    u_int8_t sctp_send_failure_event;
    u_int8_t sctp_peer_error_event;
    u_int8_t sctp_shutdown_event;
    u_int8_t sctp_partial_delivery_event;
    u_int8_t sctp_adaption_layer_event;
};
```

Subscribing Part II

- **Placing a '1' in the respective event type field turns an event on.**
- **Placing a '0' in the respective event type field turns an event off.**
- **Note that these events are the standard ones so far, other events may be added as various extensions work their way through the IETF.**

Socket Options

- **SCTP provides a host of socket options to perform a mirad of operations.**
- **Some have unique structures others just turn things on and off with boolean's or integers.**
- **SCTP_NODELAY – Turns on/off the nagel algorithm (or other delay) similar to TCP.**
- **SCTP_MAXSEG – Sets/Gets a value for the SCTP fragmentation point (an integer is passed). Note that its possible that the value the system uses is smaller than what you set.**

More Socket Options

- **SCTP_ASSOCINFO** – Retrieve or Set various information about an association. Note that not all fields in the structure are writeable.
- **SCTP_AUTOCLOSE** – Sets a idle time wherein an association will automatically close. For one-to-many style servers this can be used so that no connection state needs to be maintained by the application.
- **SCTP_ADAPTION_LAYER** – Set or Get the 32 bit adaption layer indication that will be sent with INIT's or INIT-ACK's.

More Socket Options

- **SCTP_DEFAULT_SEND_PARAM** – set or get the default sending parameters (stream number, ppid context and other fields in the `sctp_sndrcvinfo` structure).
- **SCTP_DISABLE_FRAGMENTATION** – boolean that will disable SCTP fragmentation. Note that if fragmentation is disabled, sends larger than the fragment point will be rejected with an error return code.
- **SCTP_EVENTS** – we saw this one earlier, used to set what notification events we wish to see.

More on Socket Options

- **SCTP_GET_PEER_ADDR_INFO** – get information on a peers address. The information returned includes the cwnd, srtt, rto and path mtu.
- **SCTP_I_WANT_MAPPED_V4_ADDR** – this boolean is normally on by default and makes it so an Ipv6 socket will map V4 address to V6. If this is turned off then V4 addresses will be received up a V6 socket.
- **SCTP_INITMSG** – Can be used to get or set the default INIT/INIT-ACK settings such as number of streams allowed in or requested out.

Even More on socket options

- **SCTP_PEER_ADDR_PARAMS** – allows an endpoint to get or set the heart beat interval and/or path maximum retransmits on a specific peer address.
- **SCTP_PRIMARY_ADDR** – Allows an application to specify a peer's address has the “primary” address.
- **SCTP_RTOINFO** – get or set the RTO information RTO.min, RTO.max and RTO.initial.
- **SCTP_SET_PEER_PRIMARY_ADDR** – Allows an endpoint to request that the peer change its primary address to the one specified (note this will only succeed if the peer supports the ADD-IP extension).

Final socket option page

- **SCTP_STATUS** – allows an application to retrieve a number of various parameters and stats with respect to a specific association.
- **As you can see there are a LOT of options. If you will there is a knob for about most things someone would want to do to a transport connection.**
- **The purpose of all of these knobs is to give the application better control of the transport.**
- **If you plan on using any of these options I would highly recommend getting the UNP 3rd edition. This gives all the details you will need to use these effectively (with examples).**

Extended “system calls”.

- **sctp_connectx** – Allows a user to specify multiple address to attempt to connect too.
- **sctp_bindx** – Allows an application to bind a set of addresses instead of one or all addresses.
- **sctp_opt_info** – Some implementations do not support a `getsockopt()` call that allows data to be passed both ways (some of the calls need an association id to get information). Use this call to be compatible with all implementations.

Extended “system calls”

- **sctp_getpaddr** – This call will return a block of memory holding the peers addresses currently part of the association.
- **sctp_freepaddr** – This call is used to release the memory back that the **sctp_getpaddr** call allocated.
- **sctp_getladdr** – This call will return a block of memory holding the local addresses bound to an association.
- **sctp_freeladdr** – This call should be used to release the memory allocated by **sctp-getladdr** back to the system.

Extended “system calls”

- **sctp_sendmsg** – This call will allow the caller to specify on the command line things like the stream number and other SCTPish information to be sent with a message.
- **sctp_send** – This call has a similar purpose to **sctp_sendmsg** but instead of a large number of command line options, a **sctp_sndrcvinfo** structure is used to pass the relevant information.
- **sctp_rcvmsg** – This call (as we saw previously) is used to receive a message but also a **sctp_sndrcvinfo** structure with details on the message (e.g. The stream number and stream sequence number).

Extended “system calls”

- **sctp_peeloff** – this call is used to convert a single association that is part of a one-to-many socket into an individual new socket descriptor that is a one-to-one socket.

Summary

- **SCTP is a new transport protocol**

**available now in bleeding edge Linux and BSD kernels,
and will make its way into the mainstream**

It has some cool new features

**We have given you an overview of the bits on the wire and
how it works in general**

**We have walked through how you can quickly start using
SCTP with the socket API**

- **So, where can I learn more?**

Reference Materials

- **[SCTP reference book] Stream Control Transmission Protocol (SCTP): A Reference Guide**, R. Stewart and Q. Xie, Addison-Wesley, 2002, ISBN 0-201-72186-4
- **RFC 2960: Stream Control Transmission Protocol**, October 2000
- **RFC 3309: SCTP Checksum Change**, September 2002
- **[I-G] draft-ietf-tsvwg-sctpimpguide-***: SCTP Implementer's Guide

* current version as of 21-Jul-04 is version 11

SCTP Programming References

- **[[sockets API](#)] draft-ietf-tsvwg-sctpsocket-***: **Sockets API Extensions for SCTP**
 - * current version as of 21-Jul-04 is 08
- **UNIX Network Programming, Volume 1, Third Edition**, Stevens-Fenner-Rudoff, Addison-Wesley, 2004, ISBN 0-13-141155-1

SCTP Extensions Drafts

- **[PR-SCTP]** RFC 3758
- **[Add-IP]** draft-ietf-tsvwg-addip-sctp-08: SCTP Dynamic Address Reconfiguration
- **[Pkt-Drop]** draft-stewart-sctp-pktdrprep-00: SCTP Packet Drop Reporting
- **[Auth]** draft-tuexen-sctp-auth-chunk-00: Authenticated Chunks for SCTP

Online References

- <http://www.sctp.org>

Also reachable with HTTP over SCTP!

- <http://www.ietf.org/html.charters/tsvwg-charter.html>

All current work on SCTP is done in the IETF TSVWG

- [sctp-impl](mailto:sctp-impl@mailer.cisco.com) on mailer.cisco.com



Questions?